INRIA

CENTRE DE RENNES

IRISA

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(3) 954 90 20

Rapports de Recherche

N° 215

UN PROCESSEUR POUR LA RECONNAISSANCE DE LA PAROLE

Patrice FRISON

Juillet 1983

IRIA

CENTRE DE RENNES
IRISA

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tél. (3) 954 90 20

Rapports de Recherche

N° 215

UN PROCESSEUR POUR LA RECONNAISSANCE DE LA PAROLE

Patrice FRISON

Juillet 1983

UN PROCESSEUR INTEGRE POUR LA RECONNAISSANCE DE LA PAROLE

Patrice FRISON

RESUME : Les travaux exposés dans ce rapport concernent la mise en oeuvre d'un algorithme parallèle de détection de mots dans la parole continue sur une architecture systolique intégrée. Le multiprocesseur est composé d'un réseau de processeurs élémentaires connectés de manière régulière et locale. On décrit la structure de chaque processeur et sa réalisation en circuit intégré VLSI. Le circuit comporte environ 12000 transistors. La machine complète nécessite 89 circuits et devrait permettre la détection de vocabulaires de près de 2000 mots en temps réel.

SUMMARY : The work reported here is concerned with the design of an integrated systolic architecture of a parallel word spotting algorithm for continuous speech. The multiprocessor is composed of a network of elementary processors, regularly and locally connected. We describe each processor structure and its design in VLSI. Each circuit is made out of about 12000 transistors. The whole machine uses 89 chips and should be able to achieve real time word detection in a 2000 words vocabulary.

Introduction

Les progrés considérables réalisés ces derniers temps dans le domaine de l'intégration à très grande échelle de circuits électroniques (VLSI) sont en train de modifier les rapports entre les concepteurs d'architectures et les concepteurs de circuits. En effet, il est désormais possible pour les premiers de réaliser des systèmes informatiques sur un seul circuit intégré comportant plusieurs dizaines de milliers de transistors.

Cette révolution technologique est intéressante à plusieurs titres. D'une part, l'architecte n'est plus tributaire des composants standards existants, et il lui est permis de concevoir son circuit intégré sur mesure. En particulier, il détient tous les atouts pour définir son circuit en fonction de ses besoins (fonctionnalité du circuit, vitesse, consommation, nombre de pattes du boitier etc...). D'autre part, les circuits VLSI ont un rôle important à jouer dans le domaine des architectures parallèles. Des applications peu réalistes, il y a encore quelques années, pour des raisons diverses (coût, encombrement, temps de conception), sont maintenant dans le domaine du possible.

L'une des structures parallèles les plus directement concernées par les circuits VLSI, est classée sous le terme d'architecture systolique [1]. Une machine systolique est composée d'un réseau de processeurs élémentaires, connectés régulierement et localement, à travers lequel circulent des flots de données qui interagissent lors de leur rencontre. On connait actuellement de nombreux problèmes susceptibles d'être résolus sur une architecture de ce type.

Depuis plusieurs années, nous nous sommes intéressés au problème du parallélisme à travers une application spécifique: la reconnaissance de la parole continue. En effet, les derniers développements dans ce domaine ont montré qu'on est désormais capable de concevoir des systèmes de reconnaissances performants (plus de 95% de taux de reconnaissance). Cependant, ces systèmes sont en général peu exploitables en raison de leur temps d'exécution sur calculateurs classiques. Dans le système KEAL [2], par exemple, l'une des phases de la reconnaissance qui nécessite beaucoup de temps de calculs, concerne la détection des mots. Nos travaux [3] ont porté sur une modification de l'algorithme initial qui a permis de concevoir un algorithme systolique particulierement performant. En effet, on estime pouvoir effectuer en temps reel la détection de mots appartenant à un vocabulaire d'un millier de mots.

Les travaux exposés dans ce rapport concernent la mise en oeuvre de l'algorithme parallèle de detection de mots sur une architecture systolique intégrée. Le processeur élémentaire du multiprocesseur a été conçu et les masques des circuits permettant la fabrication de la puce de silicium ont été dessinés.

Ce rapport comporte 6 parties. Le problème de la détection de mots dans un système de reconnaissance de parole est abordé dans la première partie et l'algorithme de détection est présente. La seconde partie traite de la version systolique de l'algorithme et nous montrons comment fonctionne le réseau de processeurs. La troisième partie est dédiée à la mise en oeuvre de l'algorithme sur un multiprocesseur. L'architecture globale de cette machine est décrite. On s'intéresse dans une quatrième partie à l'architecture d'un processeur élémentaire. Les différents modules constituant le processeur sont présentés. La cinquième partie nous entraine dans la structure interne des différents modules du processeurs. Enfin, nous montrons dans une sixième partie, comment on peut programmer la machine. Les programmes envisagés nous permettent de prévoir les performances de la machine.

1-L'algorithme de detection de mots

1.1-Le probleme

La reconnaissance de la parole comprend en général plusieurs étapes. La premiere consiste a digitaliser le signal de parole et a en extraire ces parametres acoustiques. Dans une seconde etape, appelee analyse phonetique, les sons elementaires du langage sont extraits et identifies. A l'issue de cette analyse, la phrase prononcee est transformee en une suite de N segments phonetiques. Notons (X(1),...,X(N)) cette sequence. Chaque segment X(i) est lui-meme constitue d'un ensemble de symboles phonetiques, qui representent les phonemes les probables pour ce segment. Dans une troisieme etape, tous les mots du vocabulaire du langage choisi sont compares a la phrase codee, chaque mot etant compare a chaque sous-suite de la phrase codee. Cette etape, appelee detection de mots, est en general tres longue.

La detection de mot consiste a rechercher dans la phrase codee, toutes les occurences d'un mot donne sous sa forme phonetique. Soit (Y(1),...,Y(M)) la suite de phonemes representant ce mot. Le probleme se resume a comparer la suite Y avec toute sous-suite (X(k+1),...,X(k+P)) extraite de X, et a evaluer la qualite de cette ressemblance.

Pour evaluer le degre de ressemblance entre un mot et une partie de la phrase codee, il doit etre tenu compte des erreurs eventuellement commises par l'analyseur phonetique. Les erreurs considerees ici sont de trois types: des confusions, des insertions et des omissions. Une confusion apparait lorsqu'un phoneme prononce z est traduit par un segment contenant plusieurs phonemes et dans lequel z peut ne pas se trouver. Une insertion se produit lorsque l'analyseur phonetique determine des segments en surnombre. Enfin un phoneme peut etre omis lorsque l'analyseur ne detecte pas le phoneme et ne construit donc pas le segment correspondant.

1.2-Un exemple

La figure 1 donne un exemple de transcription phonetique d'une phrase prononcee, en l'occurrence: "liste des connecteurs". A la suite de l'analyse phonetique, la phrase a ete scindee en 13 segments. On note que le segment X(1) a ete insere en debut de la phrase. Parmi les omissions, on note le phoneme t du mot "liste", le second phoneme k du mot "connecteur", et enfin le phoneme r de ce meme mot en fin de phrase.

Dans cette phrase codee, le detecteur de mots trouvera par exemple, le mot "liste" entre les segments X(1) et X(5), le mot "des" entre X(6) et X(7), et le mot "connecteur" entre X(8) et X(13). Mais il trouvera egalement, par exemple, les mots "piste" entre X(1) et X(5), "vecteur" entre X(10) et X(13).

1.3-L'algorithme

Le principe de l'algorithme de detection de mot est base sur un modele probabiliste de l'analyseur phonetique. Soit Y=(y(1),...,y(M)) le mot a comparer a la phrase X=(X(1),...,X(P)). Supposons que le

Comportement:	I	C	C	C	O	C	C	C	C	C	C	C	O	C	C	O
Phrase prononcee:		l	i	s	t	œ	d	e	k	o	n	ɛ	k	t	œ	r
X(i):	1	2	3	4		5	6	7	8	9	10	11		12	13	
Transcription phonetique :	p t k	l j	i e y	s f ʃ t k		o œ ɛ̃	b d g	e y ɛ	p t k	o œ ɛ̃	n v ʃ g m d z b w j	ɛ ɛ̃ e		p t k	o œ	

Figure 1: Exemple de transcription phonetique

dernier phoneme y(M) soit un symbol special] appele marqueur de fin de mot. Au mot Y est associe un automate d'etat fini probabiliste (note AEFP dans la suite du texte) qui modelise le comportement de l'analyseur phonetique lorsqu'il traite Y. La figure 2 decrit l'AEFP associe a un mot de 2 phonemes (y(1),y(2),]). Cet AEFP contient 4 etats, S(Ø),...,S(3). L'etat S(i) decrit le comportement de l'analyseur phonetique lorsqu'il traite le phoneme y(i+1). Trois possibilites s'offrent a l'analyseur:
- rester dans l'etat S(i) avec une probabilite Pi(y(i+1)) et produire un phoneme x avec une probabilite conditionnelle qi(x/y(i+1)): ce cas modelise l'insertion;
- entrer dans l'etat S(i+1) avec la probabilite Pc(y(i+1)) et produire un phoneme x avec un probabilite conditionnelle qc(x/y(i+1)): c'est le cas de la confusion;
- entrer dans l'etat S(i+1) avec la probabilite Po(y(i+1)), sans produire de phoneme: c'est le cas de l'omission.

Il est clair que, dans le cas du marqueur de fin de mot, Pi(])=Pc(])=Ø, afin d'eviter que l'AEFP ne produise le marqueur. Supposons que chaque segment X(j) ne contient qu'un seul phoneme x(j), il a ete montre [4] que la probabilite que Y ait ete prononce est egale a la probabilite que l'AEPF associe a Y produise la suite de phonemes x(1),...,x(P) en transitant de S(Ø) a S(M+1). Soit L(i,j) la probabilite pour l'AEFP d'entrer dans l'etat S(i) apres avoir produit x(1),...,x(j), alors L est caracterise par les formules suivantes:

(1) $L(i,j) = Lc(i,j)+Li(i,j)+Lo(i,j)$

avec

(2) $Lc(i,j)= L(i-1,j-1)xPc(y(i))xqc(x(j)/y(i))$

(3) $Li(i,j)= L(i,j-1)xPi(y(i+1))xqi(x(j)/y(i+1))$

(4) $Lo(i,j)= L(i-1,j)xPo(y(i))$

Lc(i,j), Li(i,j) et Lo(i,j) denotent respectivement les probabilites pour l'AEFP d'entrer dans l'etat S(i) apres confusion de x(j) et y(i), insertion de x(j) ou omission de y(i). Les equations (1) a (4) sont valides pour 1<=i<=M et 1<=j<=P, avec les conditions initiales suivantes:

(2.1) $(\forall i)(1<=i<=M)\ Lc(i,Ø)=Ø$,
$(\forall j)(1<=j<=P)\ Lc(Ø,j)=Ø$,
$Lc(Ø,Ø)=1$

(3.1) $(\forall i)(Ø<=i<=M)\ Li(i,Ø)=Ø$

(4.1) $(\forall j)(Ø<=j<=P)\ Lo(Ø,j)=Ø$

et finalement, pour completer le schema de recurrence:

(5) $(\forall j)(Ø<=j<=P)\ L(M+1,j)=Lo(M+1,j)=L(M,j)$

qui derive du fait que Pc(])=Ø, Po(])=1 et que S(M+1) n'a pas de transition d'insertion.

Dans le cas de segments contenant plusieurs noms de phonemes, les equations (2) et (3) doivent etre modifiees. Si l'on suppose que chaque segment X(j) contient n phonemes notes x(j,1),...,x(j,n) avec

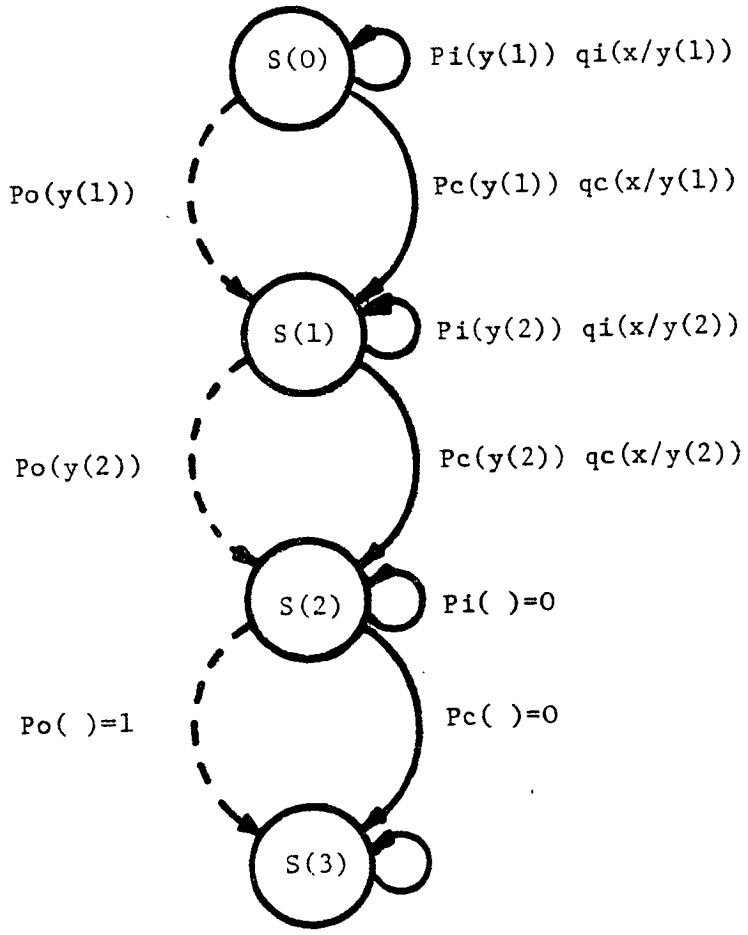


Figure 2: l'AEPF associe au mot y(1).y(2)

des probabilites respectives p(j,1),...,p(j,n), les equations (2) et (3) sont modifiees en remplacant les termes qc(x/y) et qi(x/y) par leur valeur moyenne sur X(j), soient:

$$(6) \quad Lc(i,j) = L(i-1,j-1) \times Pc(y(i)) \times \sum_{k=1}^{n} p(j,k) \times qc(x(j,k)/y(i))$$

$$(7) \quad Li(i,j) = L(i,j-1) \times Pi(y(i+1)) \times \sum_{k=1}^{n} p(j,k) \times qi(x(j,k))/y(i+1))$$

1.4-Application a la detection de mots dans un phrase

Soit V={Y(1),...,Y(d)} un vocabulaire de d mots a reconnaitre et soit X=(X(1),...,X(N)) la transcription phonetique d'une phrase prononcee. Le probleme est de comparer chaque mot Y de V a chaque sous-suite (X(k+1),...,X(k+p)) de X (Ø<=k<=k+p<=N). Chacune de ces comparaisons fournit une vraisemblance L. Si L est une valeur suffisamment bonne (par exemple, si L depasse un certain seuil), on dit que Y est detecte entre les segments X(k+1) et X(k+p).

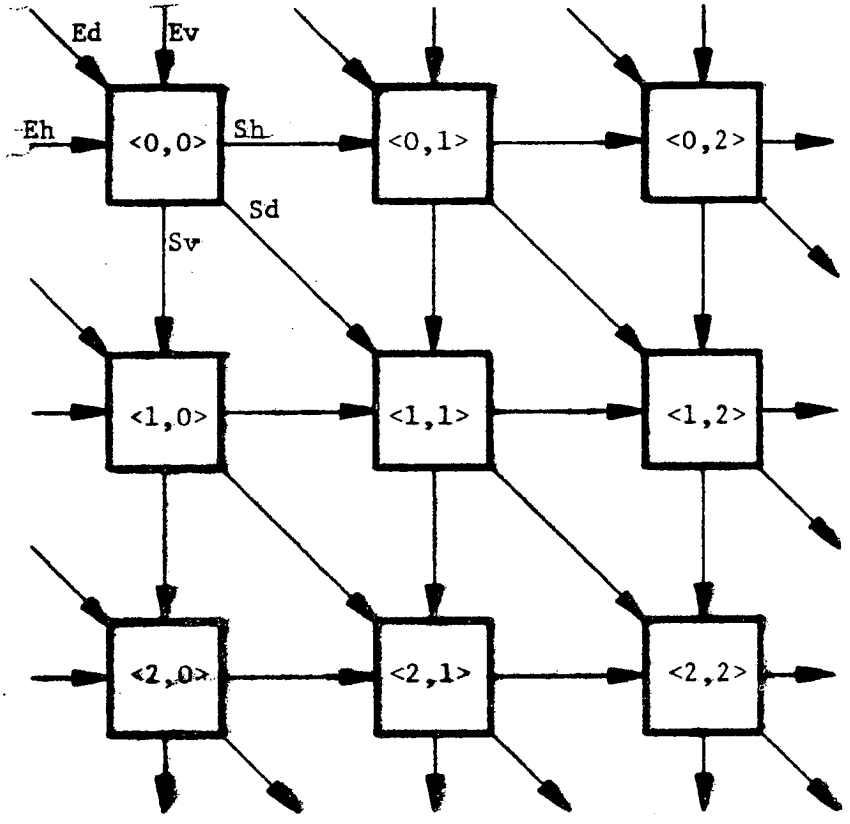


Figure 3: structure du reseau

2-Description du reseau de processeurs

2.1-Le reseau

L'idee de base consiste a attacher un processeur a chaque point (i,j). Notons <i,j> le processeur associe a (i,j). <i,j> recoit les valeurs Lc(i,j), Lo(i,j) et Li(i,j). Il calcule d'abord L(i,j) en appliquant l'equation (1) puis il calcule Lc(i+1,j+1), Li(i,j+1) et Lo(i+1,j) en appliquant respectivement les equations (6),(7) et (4). Pour effectuer ces calculs, <i,j> doit recevoir egalement les valeurs X(j+1) et y(i+1). En consequence, le reseau est forme de (M+1) x (P+1) processeurs <i,j> avec Ø<=i<=M et Ø<=j<=P. <i,j> doit etre connecte a <i-1,j-1>, <i-1,j> et <i,j-1> pour pouvoir calculer L(i,j). La stucture du reseau est representee sur la figure 3.

Cette structure montre que chaque processeur possede 6 ports d'entrees/sorties (3 ports d'entrees et 3 ports de sorties). Les lignes de connexion entre processeurs sont unidirectionnelles (leur direction est indiquee par les fleches). Pour chaque processeur les ports d'entrees/sorties sont notes comme suit: Ed (entree diagonale), Ev (entree verticale), Eh(entree horizontale) pour les entrees et Sd (sortie diagonale), Sv (sortie verticale), Sh (sortie horizontale) pour les sorties.

2.2-Fonctionnement du reseau

On examine tout d'abord le cycle de base d'un processeur, puis on montre comment le reseau peut etre utilise pour detecter un mot Y dans la suite de segments X. Enfin on decrit deux modes de fonctionnement pipeline du reseau qui permettent de resoudre completement le probleme.

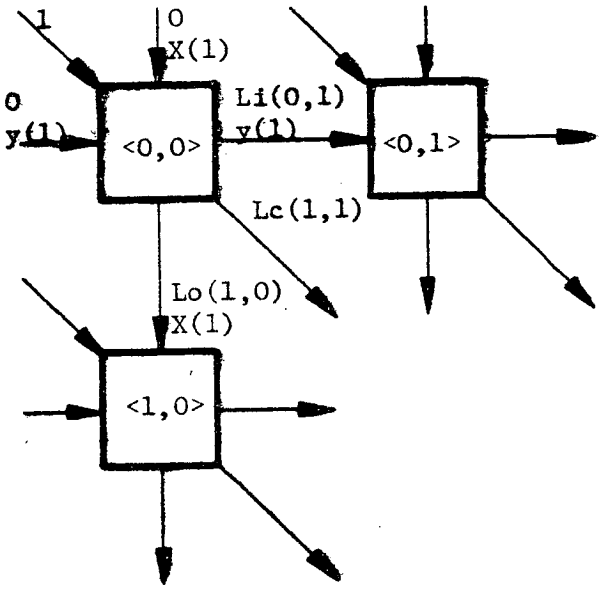
2.2.1-Cycle de base d'un processeur

Considerons le processeur <i,j> et supposons que les donnees suivantes se trouvent disponibles sur ces ports d'entrees:
- sur Ev, successivement X(j+1) et Lo(i,j)
- sur Ed, Lc(i,j)
- sur Eh, successivement Y(i+1) et Li(i,j)

Pendant son cycle de base, le processeur calcule L(i,j) par application de l'equation (1) puis calcule Li(i,j+1), Lc(i+1,j+1), et Lo(i+1,j) en appliquant respectivement les equations (7), (6) et (4). Pour se faire, le processeur memorise d'une facon permanente les distributions de probabilite qc(x/y), Pi(y) et Po(y). Le processeur <i,j> emet ensuite les valeurs suivantes:
- y(i+1) et L(i,j+1) sur Sh;
- Lc(i+1,j+1) sur Sd;
- X(j+1) et Lo(i+1,j) sur Sv.

Ce traitement est valide pour les processeurs interieurs i.e. les processeurs <i,j> satisfaisant 1<=i<=M et 1<=j<=P. Les remarques suivantes montrent que le traitement decrit precedemment est egalement valide pour les processeurs des bords.
- les equations (2.1) et (3.1) definissent les valeurs initiales a envoyer aux processeurs <i,j> avec i=Ø et i<=j<=P-1 ou j=Ø et 1<=i<=M.

ETAPE 0



ETAPE 1

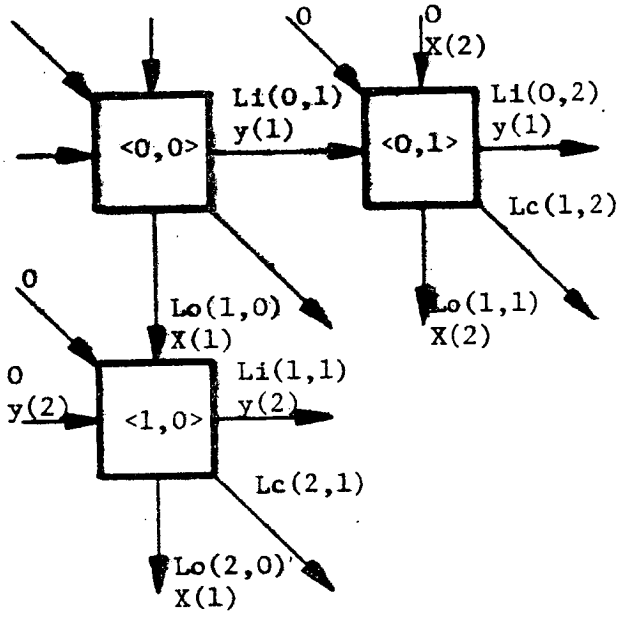


Figure 4: fonctionnement du reseau

- les processeurs <M,j> avec 1<=j<=P-1, produisent L(M+1,j) sur leur port Sv, puisque Pc(])=Ø.
- enfin, considerons les processeurs <i,P> ou O<=i<=M. Afin que les processeurs se comportent exactement comme les autres, il est pratique d'introduire un segment fictif de fin de phrase X(P+1) contenant uniquement le marqueur de fin de mot]. Si l'on pose:

(8) (∀y) qc(]/y)=qi(]/y)=Ø

on montre que <i,P> delivre Lo(i+1,P) sur son port Sv et donc que <M,P> delivre L(M+1,P)=Lo(M+1,P) sur son port Sv.

2.2.2-Detection d'un mot

Supposons que le reseau fonctionne de facon synchrone. Aux instants successifs Ø,1,...,t,.. chaque processeur execute le calcul precedant. On veut utiliser le reseau pour detecter le mot Y=(y(1),...,y(M)) dans la sequence X=(X(1)...X(P)). La figure 4 illustre le fontionnement du reseau

Au temps t=Ø, le processeur <Ø,Ø> recoit X(1) et Ø sur Ev, 1 sur Ed et y(1) et Ø sur Eh. <Ø,Ø> execute son programme puis emet X(1) et Lo(1,Ø) sur Sv, Lc(1,1) sur Sd et y(1) et Li(Ø,1) sur Sh.

Au temps t=1, <1,Ø> recoit les valeurs envoyees par <Ø,Ø> sur Ev, Ø sur Ed, y(2) et Ø sur Eh. Il envoie ensuite X(1) et Lo(2,Ø) sur Sv, Lc(2,1) sur Sd et y(2) et Li(1,1) sur Sh. Au meme moment , <Ø,1> recoit X(2) et Ø sur Ev, et Ø sur Ed et y(1) et Li(Ø,1) sur Eh. Il produit ensuite X(2) et Lo(1,1) sur Sv, Lc(1,2) sur Sd, Y(1) et Li(Ø,2) sur Sh.

Plus generalement, au temps t, X(t+1) et Ø sont disponibles sur l'entree Ev du processeur <Ø,t> et Ø sur l'entree de <Ø,t>. De meme , Y(t+1) et Ø sont envoyes sur Eh de <t,Ø> et Ø sur Ed de <t,Ø>. La diagonale de processeurs <i,j> avec i+j=t, delivre ensuite X(i+1) et Lo(i+1,j) sur Sv, Lc(i+1,j+1) sur Sd et enfin y(i+1) et Li(i,j+1) sur Sh.

Au temps t=M+P, on voit que la probabilite L(M+1,P) de correspondance entre X et Y apparait sur le processeur <M,P>. On peut noter que le reseau fonctionne egalement pour la detection de mots de longueur l inferieure a M, a condition que ces mots soient completes par M-l marqueurs de fin de mot. De meme , si la phrase a une longueur l inferieure a P, la transcription phonetique de cette phrase doit etre completee par P-l marqueurs de fin de phrase.

2.2.3-Fonctionnement du reseau en mode pipeline

Supposons tout d'abord que les mots du vocabulaire ont une longueur inferieure ou egale a M. Nous avons vu qu'a l'instant t, seuls les processeurs <i,j> avec i+j=t sont actifs. Ceci suggere une meilleure methode d'introduction des donnees dans le reseau afin d'effectuer des traitements a la chaine (ou pipeline). Deux modes de traitement pipeline sont possibles : le pipeline sur les entrees y(i) (appele aussi pipeline sur les mots) et le pipeline sur les entrees X(j) (appele aussi pipeline sur la phrase).

2.2.3.1-Pipe-line sur les mots

En fixant sur les entrees verticales du reseau les valeurs X(1),...,X(P) et en presentant successivement les mots Y(Ø),...,Y(d) sur les entrees horizontales, il est possible de comparer consecutivement chaque mot du vocabulaire V, avec la meme sequence X(1),...,X(P). Ce mode de fonctionnement est illustre sur la figure 5.

A chaque cycle, le premier segment Y(1,1) d'un nouveau mot Y(1) est introduit dans le reseau vers le processeur <Ø,Ø>. Les cycles suivants, les segments Y(1,2),Y(1,3),...,Y(1,M+1) sont envoyes respectivement sur <1,Ø>,<2,Ø>,...,<M,Ø> a raison de 1 par cycle processeur.

2.2.3.2-Pipe-line sur la phrase

En fixant le mot Y=(y(1),...,Y(M)) sur les entrees horizontales et en decalant les segments de la phrase de gauche a droite sur les entrees verticales, il est possible de comparer successivement chaque sous-sequence de X(1),...,X(t) avec Y. La figure 6 illustre ce principe.

2.2.3.3-Combinaison des deux modes de pipeline

En combinant ces deux modes de pipeline, il est possible de detecter tous les mots de V dans une phrase X(1),...,X(N). L'une des formes les plus naturelles consiste a comparer tous les mots du vocabulaire aux sous-suites commencant a X(1), puis aux sous-suites commencant a X(2) et ainsi de suite. En effet, si l'on desire reconnaitre de la parole en temps reel, ce mode de combinaison des deux pipelines, permet le lancement du processus de detection des que les premiers segments de X sont produits par l'analyseur phonetique, le traitement s'operant au fur et a mesure de leur production. C'est ce mode de fonctionnement que l'on considere dans la suite du texte.

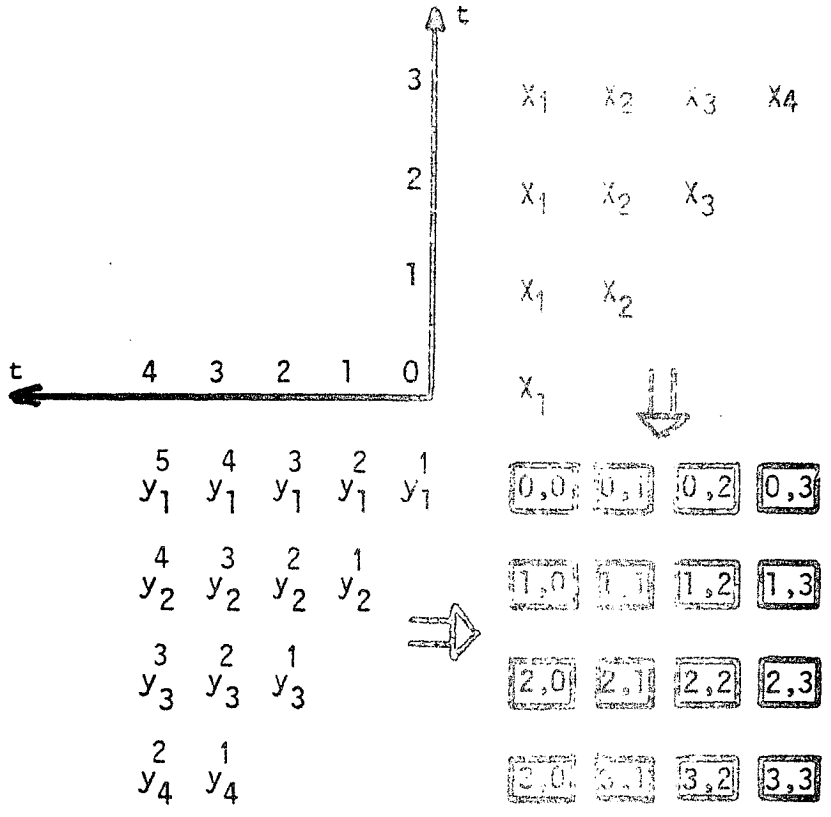


Figure 5: fonctionnement du reseau en mode pipeline sur les mots

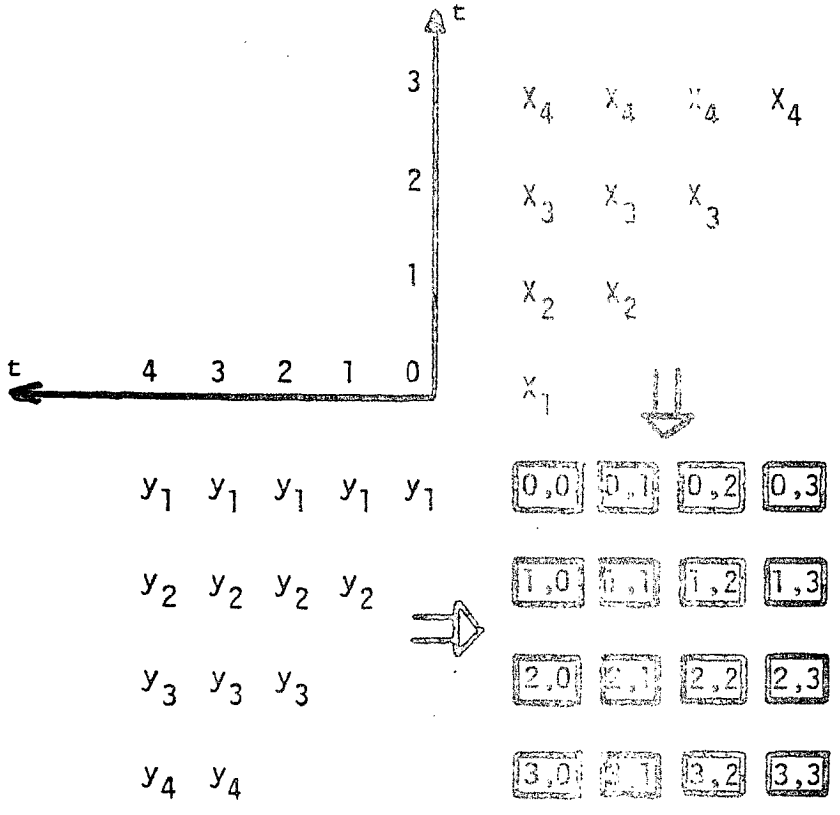


Figure 6: fonctionnement du reseau en mode pipeline sur la phrase

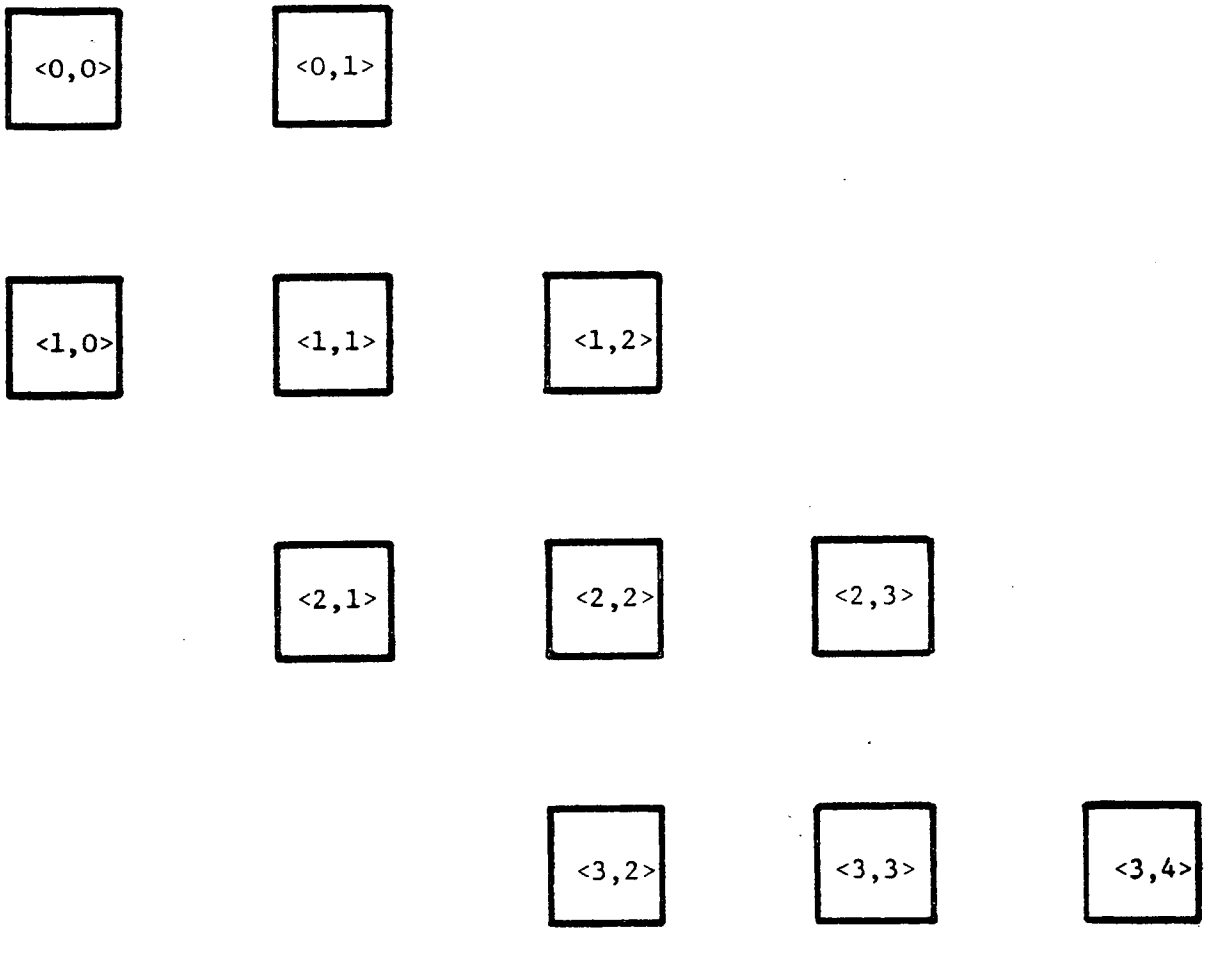


Figure 7: forme du reseau pour M=3 et delta=1

3-Mise en oeuvre et architecture du reseau

Le but de ce paragraphe est de decrire l'architecture de processeurs capable de supporter la structure multiprocesseur presentee dans le paragraphe precedant. Trois points seront successivement abordes: le nombre de processeurs, le controle du systeme et les connexions entre processeurs.

3.1-Taille du reseau

Si l'on suppose que toutes les correspondances entre un mot de longueur M et une phrase de longueur N sont vraisemblables, le nombre de processeurs est (M+1)x(N+1). Dans la pratique, on constate que les seules valeurs L(i,j) "proches" de la diagonale i=j sont significatives. Il est habituel de ne considerer que les valeurs (i,j) pour lesquelles:

abs(i-j) <= delta

ou delta est une constante. Cela conduit a donner au reseau une forme telle que celle de la figure 7.

Les valeurs X(j) et Y(i) peuvent etre entrees directement sur les processeurs des bords de la diagonale a condition de leur appliquer un retard approprie. Les formules (3.1) fournissant les valeurs initiales des termes Li et Lo doivent etre modifiees de la maniere suivante:

(3.2) (∀i),(∀j), (i-j>=delta) Li(i,j)=Ø

(∀i),(∀j), (j-i>=delta) Lo(i,j)=Ø

Un calcul simple montre alors que le nombre de processeurs du reseau est:

Np=(M+1)x(2xdelta+1)-deltax(delta+1)/2

A titre indicatif, M peut etre fixe a 1Ø et delta a 4 ce qui donne Np=89 processeurs.

3.2-Controle du systeme

Comme il a ete vu precedemment, le cycle de base d'un processeur se decompose en 3 parties: reception des donnees sur les ports d'entrees, calcul des valeurs Lc, Lo, Li et envoie des donnees sur les ports de sorties. Les processeurs peuvent fonctionner independemment, en supposant par exemple que les communications interprocesseurs se font suivant un mecanisme de producteur-consommateur. Ceci entraine une structure de machine tres generale de type MIMD.

Or, il a ete montre que lors du traitement en mode pipeline sur les mots, les processeurs d'une meme diagonale traitent un meme mot et de plus, ils effectuent le meme programme (cycle de base). Cette remarque suggere le controle des processeurs d'une meme diagonale par un controleur unique. Celui-ci envoie un seul flot d'instructions executees simultanement par tous les processeurs d'une meme diagonale (mode de fonctionnement SIMD).

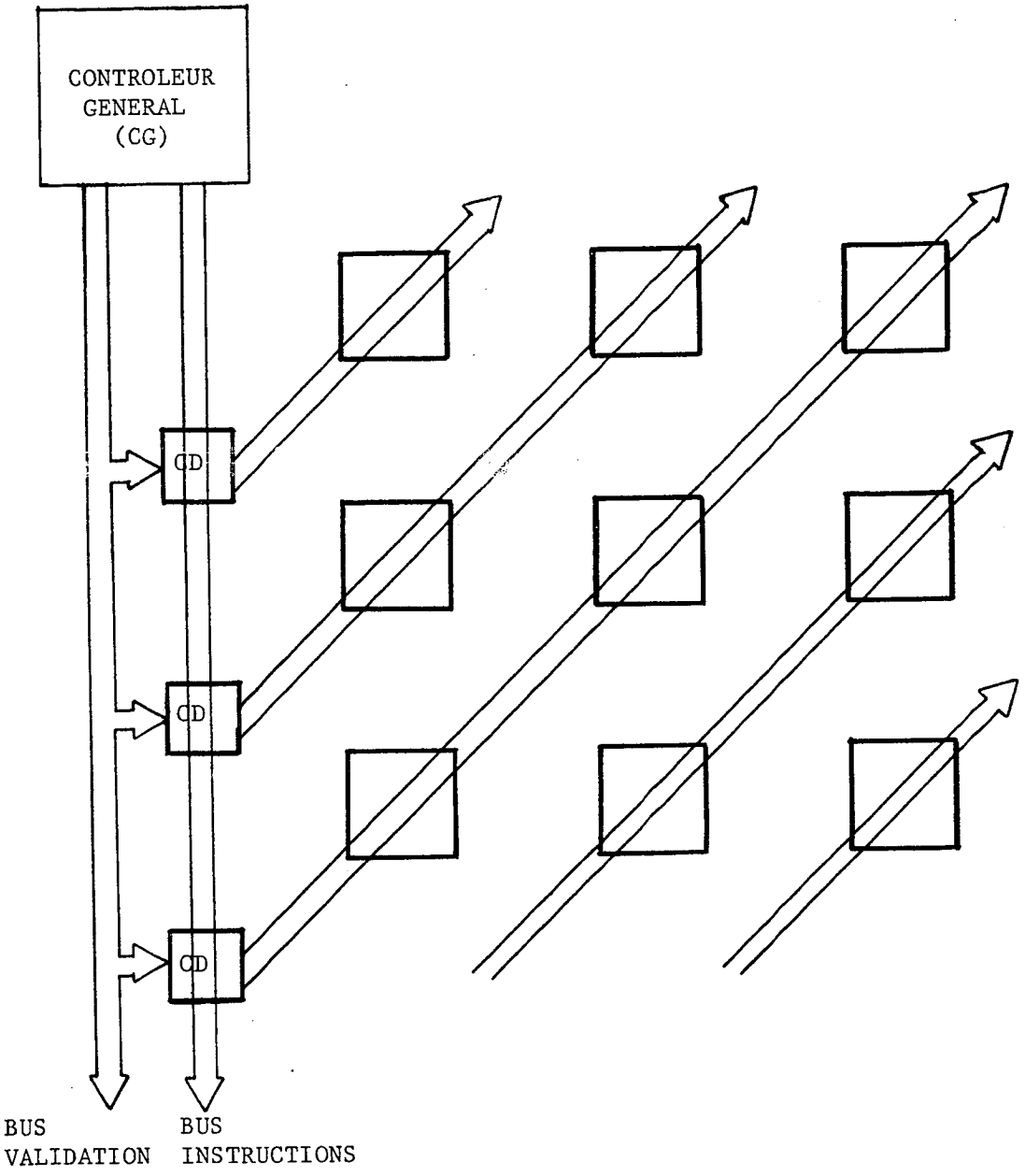


Figure 8: Organisation du controle de la machine

Par ailleurs, il est clair que les processeurs de deux diagonales successives executent le meme programme avec un certain decalage. En fixant ce decalage a un cycle processeur exactement, il est alors possible de controler les deux diagonales avec le meme controleur. Par consequent, on en deduit qu'un seul controleur (appele controleur general et note CG) est utilisable pour tout le reseau, tous les processeurs executant la meme instruction simultanement avec leur donnees propres. Cependant, il est necessaire de pouvoir inhiber certaines diagonales afin que les processeurs qui les composent soient inactifs, par exemple, lors du debut ou de la fin du pipeline sur les mots.

Dans ce but, a chaque diagonale est associee un controleur de diagonale (note CD). Ce controleur peut se trouver dans deux etats. Dans l'etat actif, il transmet aux processeurs de sa diagonale, les instructions qui lui sont envoyees par le controleur general. Dans l'etat inactif, il inhibe l'instruction du controleur, de telle sorte que les processeurs de la diagonale n'effectuent aucune operation. La figure 8 montre l'organisation du reseau de processeurs et de ces controleurs.

Deux bus sont controles par le controleur general: le bus d'instructions, sur lequel est codee l'instruction a executer et le bus de validation qui sert a activer ou desactiver un controleur de diagonale.

3.3-Connexions interprocesseurs

La struture du reseau necessite theoriquement que chaque processeur possede 6 ports d'entrees-sorties, (3 d'entrees et 3 de sorties). En realite, il est suffisant de ne mettre en oeuvre que 3 connecteurs d'entrees-sorties par processeur. Deux connecteurs d'entrees et un connecteur de sortie. La figure 9, montre comment les processeurs sont interconnectes physiquement.

Chaque processeur possede deux ports d'entrees notes V (entree verticale) et H (entree horizontale), et un port de sortie note S. Nous allons montrer comment cette organisation permet de realiser les differents transferts de donnees interprocesseurs necessites par l'algorithme.

3.3.1-Transferts horizontaux et verticaux

Lorsqu'un processeur lit une donnee de type t, sur son port d'entree verticale (resp horizontale), le fonctionnement synchrone du reseau entraine que tous les processeurs lisent simultanement sur leur port V (resp port H) une donnee de type t et que, par consequent, tous les processeurs ont affiche sur leur port de sortie la valeur de type t qu'ils viennent d'utiliser ou de calculer.

3.3.2-Transferts diagonaux

Dans le cas d'un transfert logique diagonal, on constate que l'information recue par un processeur doit etre memorisee pendant un cycle processeur avant de pouvoir etre utilisee. En effet, si un processeur emetteur appartient a la diagonale d, le processeur receveur

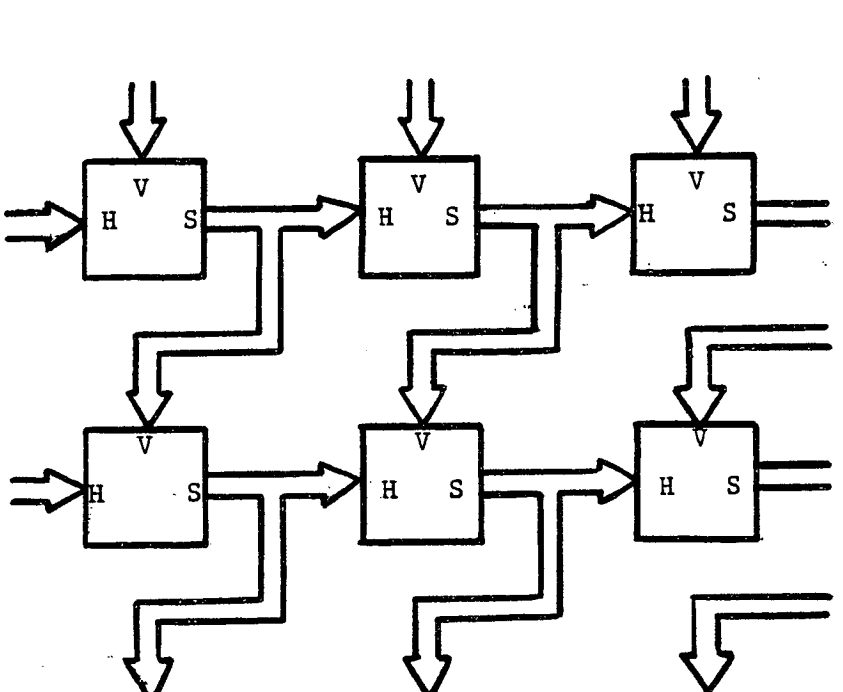


Figure 9: connexions interprocesseurs

appartient a la diagonale d+2 (voir figure 3). Un transfert logique diagonal sera remplace par deux transferts physiques: un transfert vertical, puis un transfert horizontal.

4-Architecture d'un processeur

Dans le travail de conception de l'architecture d'un processeur, en vue de son integration, il faut essayer de calquer l'architecture du processeur sur l'algorithme qu'il doit effectuer, en ce sens que seules les fonctions necessaires doivent etre mises en oeuvre. Par ailleurs le processeur doit rester aussi simple que possible. Dans cette partie, on introduit les differentes parties du processeur. Dans un premier paragraphe nous reprenons en detail le programme a effectuer par un processeur, puis l'architecture globale qui en resulte est decrite dans un second paragraphe. Enfin, les differents modules du processeur sont detailles dans un dernier paragraphe.

4.1-Le programme de base du processeur

Afin de definir les modules constituants le processeur, il est necessaire de preciser les fonctions qu'il doit realiser et donc definir precisemment le programme qu'il doit effectuer. Pendant son cycle de base, le processeur <i,j> calcule tout d'abord L(i,j) par application de l'equation (1) c'est a dire:

(1) $L(i,j)=Lc(i,j)+Li(i,j)+Lo(i,j)$

puis calcule Li(i,j+1), Lc(i+1,j+1) et Lo(i+1,j) par application des equations (7), (6) et (4) soit:

(9) $Li(i,j+1)=L(i,j)\times Pi(y(i+1)\times\sum_{k=1}^{n}p(j+1,k)\times qi(x(j+1,k)/y(i+1))$

(1Ø) $Lc(i+1,j+1)=L(i,j)\times Pc(y(i+1))\times\sum_{k=1}^{n}p(j+1,k)\times qc(x(j+1,k)/y(i+1))$

(11) $Lo(i+1,j)=L(i,j)\times Po(y(i+1))$

Avant de pouvoir decrire completement le programme du processeur, il nous faut definir la valeur de n (n indique le nombre de phonemes engendres par l'analyseur phonetique pour chaque segment de parole X(i)). La valeur 3 a ete choisie pour n car elle correspond a la valeur generalement choisie.

Supposons que le processeur ait recu, de ses voisins, toutes les informations necessaires, et que ces valeurs soient rangees dans les variables suivantes:

PX1,PX2,PX3: p(j+1,1),p(j+1,2),p(j+1,3)
Yi,Yc,Yo : Pi(y(i+1)),Pc(y(i+1)),Po(y(i+1))
X1,X2,X3 : nom des phonemes x(j+1,1),x(j+1,2),x(j+1,3)
Y : nom du phoneme y(i+1)
Qc,Qi : tableaux contenant respectivement les probabilites de confusions et d'insertions qc(x/y) et qi(x/y)

Par ailleurs, en supposant que les valeurs L,Lc,Li,Lo sont calculees dans des variables de meme nom, le programme 1 suivant se deduit des equations (1), (9), (1Ø) et (11).

```
debut
   L:=Lc+Li+Lo;
   Li:=LxYix(PX1xQi[X1,Y]+PX2xQi[X2,Y]+PX3xQi[X3,Y]);
   Lc:=LxYcx(PX1xQc[X1,Y]+PX2xQc[X2,Y]+PX3xQc[X3,Y]);
   Lo:=LxYo
fin
```

Programme 1

Afin de simplifier l'architecture du processeur, deux modifications ont ete apportees au programme 1. L'une concerne la taille des tableaux a memoriser dans le processeur, la seconde est liee a la representation des donnees.

4.1.1-Reduction de la taille des tableaux

Le programme 1 indique que 2 tableaux Qc et Qi doivent etre memorises en permanence dans le processeur car ils contiennent des informations propres au systeme de reconnaissance et sont donc utilisables pendant tout le traitement d'une phrase. Soit F le nombre total de phonemes du langage, exception faite du marqueur], la taille de Qc et Qi est donnee par (F+1)x(F+1). Pour fixer les idees, en posant F=32 et le nombre de bits par mot egal a 1Ø, la taille cumulee des 2 tableaux vaut plus de 21k bits. Si l'on fait l'hypothese que ces memoires sont des memoires vives, chargees dynamiquement en fonction du langage choisi ou du locuteur par exemple, cette taille de memoire vive est trop importante pour que l'on puisse envisager l'integration d'un processeur complet sur une puce de silicium, dans la technologie visee (NMOS 5 microns).

Cependant, dans le cas du traitement en mode pipeline sur les mots illustre par la figure 5, il apparait que les processeurs d'une meme colonne j recoivent le meme segment X(j+1) pendant que tous les mots du vocabulaire defilent sur les entrees horizontales du reseau. Cette remarque suggere de ne memoriser dans les processeurs que les valeurs qc(x/y) et qi(x/y) pour toutes les valeurs de y mais pour 3 valeurs de x seulement, puisqu'un segment X(j) contient 3 noms de phonemes (n=3).

Par ailleurs, nous avons fait l'hypothese que les probabilites conditionnelles qi(x/y) etaient independantes de y. Finalement, le tableau Qc se reduit a un tableau de 3x(F+1) elements et le tableau Qi ne contient plus que 3 elements seulement.

4.1.2-Representation des valeurs

Les calculs effectues par le processeur etant, en general, des produits de probabilites, les resultats s'etendent sur une large echelle entre Ø et 1. Par consequent, l'utilisation d'operations en virgule fixe est peu realiste. Par ailleurs, l'utilisation d'operations en virgule flottante risque d'entrainer de nombreux problemes. La complexite des circuits, la taille du microprocesseur, la vitesse d'execution des operations en sont quelques exemples. L'utilisation des logarithmes permet de resoudre, en partie, ces problemes (dans la suite du texte, les logarithmes a base 2 sont consideres).

Les multiplications, operations les plus utilisees, sont remplacees par des additions, qui ont l'avantage de s'effectuer rapidement et d'etre faciles a mettre en oeuvre.

Les additions, par contre, sont plus difficiles a mettre en oeuvre. La methode utilisee pour calculer Log(a+b) en fonction de Log(a) et Log(b) est la suivante. Sachant que:

(12) $Log(a+b)=Log(a)+Log(1+b/a)$
$=Log(b)+Log(1+a/b)$

le probleme se reduit a calculer la valeur Log(1+u) connaissant Log(u). Posons Z la fonction telle que:

(13) $Z: t \dashrightarrow Z(t)=Log(1+2^t)$

l'equation (12) devient:

(14.1) $Log(a+b)=Log(a)+Z(Log(b)-Log(a))$
(14.2) $=Log(b)+Z(Log(a)-Log(b))$
(14.3) $=max(Log(a),Log(b))+Z(-abs(Log(a)-Log(b)))$

Les equations (14.1) et (14.2) montrent que Log(a+b) peut etre calcule de 2 facons differentes. La formule (14.3) tire profit de ce resultat et montre, en particulier, que l'espace de depart de la fonction Z peut etre reduit a l'ensemble des valeurs negatives. Cette remarque est tres importante, car comme nous le verrons par la suite, la fonction Z sera tabulee.

Le programme 2 suivant, est la version modifiee et simplifiee du programme 1. Les variables T1, T2 et T3 ont ete introduites pour memoriser certains resultats intermediaires. La fonction SOMLOG, recoit deux parametres en entree: P1=Log(a) et P2=Log(b) puis calcule Log(a+b).

```
debut
{ calcul de L }
   L:=SOMLOG(SOMLOG(Li,Lc),Lo);
{ calcul de Li }
   T1:=PX1+Qi[1];
   T2:=PX2+Qi[2];
   T3:=PX3+Qi[3];
   Li:=L+Yi+SOMLOG(SOMLOG(T1,T2),T3);
{ calcul de Lc }
   T1:=PX1+Qc[1,Y];
   T2:=PX2+Qc[2,Y];1
   T3:=PX3+Qc[3,Y];
   Lc:=L+Yc+SOMLOG(SOMLOG(T1,T2),T3);
{ calcul de Li }
   Lo:=L+Yo;
fin
```

Programme 2

4.1.3-Codage des valeurs

Il a ete choisi de coder les logarithmes des probabilites sur 16 bits en complement a 2, afin d'une part, de conserver une bonne

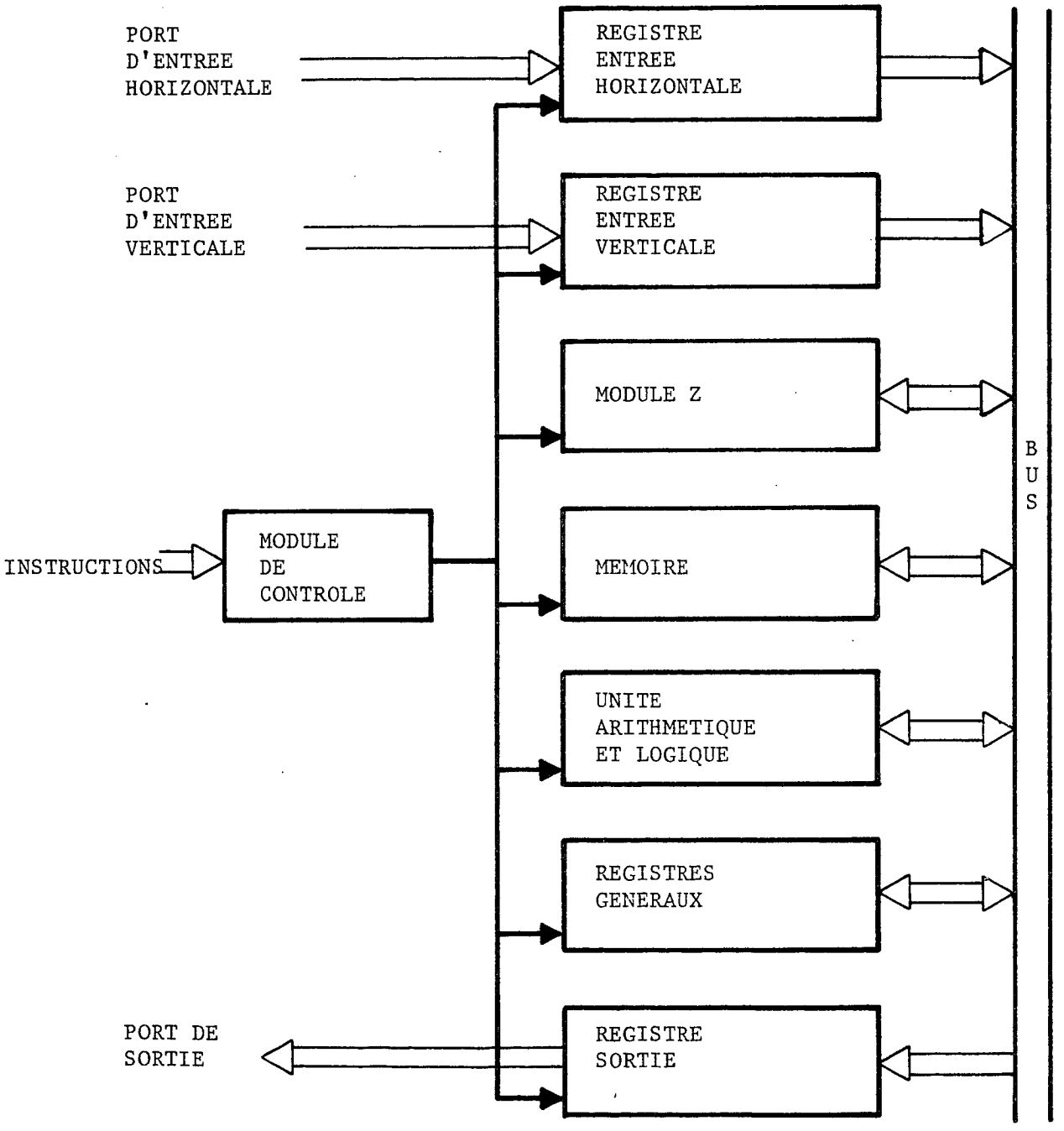


Figure 10: architecture d'un processeur

precision sur les calculs, et d'autre part, de coder les valeurs sur une large etendue. Un mot de 16 bits est compose de 2 parties. Les 9 bits de poids forts constituent la partie entiere de la valeur; les 7 bits de poids faibles forment la partie fractionnaire. Ceci permet le codage des probabilites avec une bonne precision puisque 2 valeurs codees consecutives u et v (u<v) representent 2 probabilites pu et pv dont le rapport pu/pv est constant et vaut Ø.9946. A noter que la valeur de probabilite Ø n'est pas codable, elle est cependant approchee par la valeur notee Min qui vaut 2**-256, soit environ 1Ø**-77.

4.2-Structure interne du processeur

Le processeur est constitue de 4 modules principaux:
- un module memoire ou sont stockees les valeurs de probabilites de confusion Qc(x/y);
- une unite arithmetique et logique capable d'effectuer des operations elementaires telles que soustraction, addition, incrementation;
- un tableau de registres generaux servant de memoire de travail;
- un module, note Z, utilise pour tabuler la fonction Z.

Ces modules sont organises autour d'un bus unique comme l'illustre la figure 1Ø. Egalement connectes au bus, se trouvent 3 registres specialises, notes RS (registre sortie), RH (registre entree horizontale), RV (registre entree verticale). Ils permettent au processeur de communiquer avec l'exterieur.

Un module de controle fait egalement partie du processeur. Son role est de recevoir les instructions envoyees par le controleur CG, de les decoder et d'envoyer les commandes ainsi engendrees vers les differents modules.

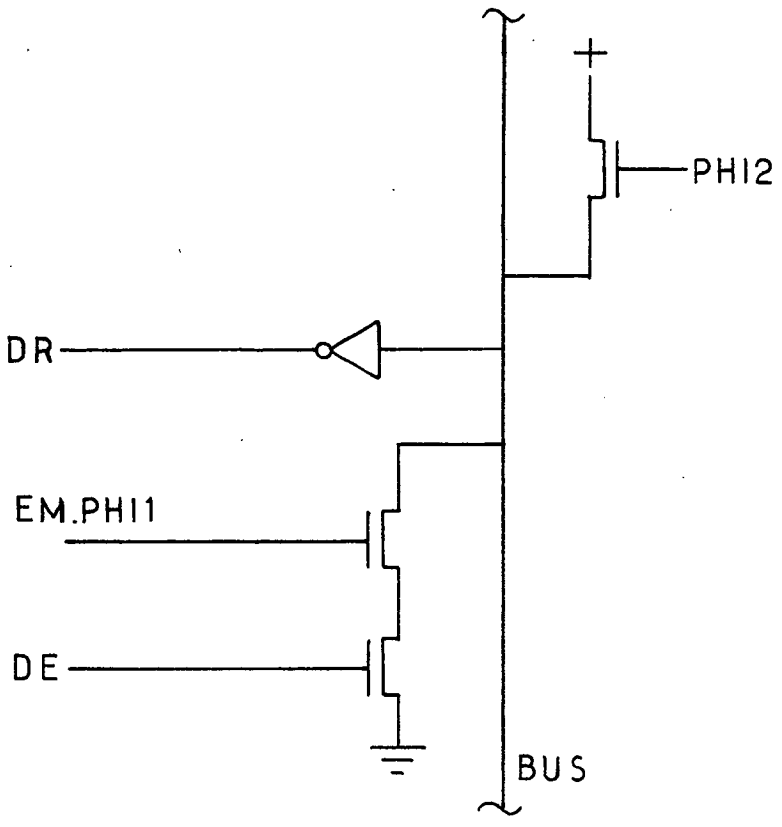


Figure 11: principe de fonctionnement du bus

5-Structure interne des modules du processeur

Dans cette partie, les differents modules constituants le processeur sont etudies en detail. La technologie choisie est la technologie NMOS. Les circuits sont synchronises par une horloge. L'horloge est composee de deux phases non recouvrantes notees PHI1 et PHI2. Pour des raisons de typographie, nous noterons Xbar le complement d'un signal X.

On etudie dans un premier paragraphe, la structure du bus sur lequel sont connectes la plupart des modules. Dans les paragraphes suivants, on etudie successivement la memoire, le module Z, l'unite arithmetique et logique, le tableau de registres et enfin, le module de controle.

5.1-Le bus

Le bus permet le transfert d'information d'une unite a l'autre. Deux criteres ont preside a sa conception. D'une part, il doit etre rapide et d'autre part, il doit pouvoir fonctionner avec un nombre quelconque de modules. Son fonctionnement repose sur la technique du prechargement: pendant la phase PHI2, le bus est precharge, puis pendant la phase PHI1 suivante, l'information a transmettre est envoyee sur le bus. Le principe de fonctionnement du bus est illustre sur la figure 11.

Pendant PHI2, le bus, agissant comme une capacite, est charge. Puis pendant PHI1, si la commande d'emission (notee EM) est active, la donnee a emettre (notee DE) produira l'un des effets suivants. Si DE vaut 1, le bus est decharge; si DE vaut Ø, le bus reste charge. Il apparait donc que le bus contient l'information DE complementee (BUS=DEbar). Par consequent, la donnee recue par un module (notee DR), doit etre egale a la donnee du bus complementee (DR=BUSbar).

Cette structure de bus repond aux specifications citees plus haut:
- la rapidite est assuree. En effet, dans le cas le plus defavorable ou l'information du bus doit etre Ø, cette valeur est valide tres peu de temps apres le debut de PHI1 car le dechargement du bus est une operation tres rapide.
- il est clair que cette organisation permet la connexion d'un nombre quelconque de modules sur le bus.

5.2-La memoire

Elle contient les valeurs de probabilites de confusion qc(x/y) pour 3 valeurs de x et F (nombre de phonemes) valeurs de y. Cependant, on remarque que le tableau Qc est relativement creux puisque Qc[i,j]=Ø si i et j ne sont pas des phonemes de meme type (consonne ou voyelle). Ceci suggere une organisation particuliere de la memoire afin d'exploiter ce phenomene. La memoire est adressee par un triplet (A,B,C) defini de la facon suivante:
- A prend ces valeurs dans l'intervalle [Ø,19] et indique le numero d'un phoneme y dans son type. Les phonemes se repartissent en 12 voyelles et 2Ø consonnes dans la langue Francaise.
- B prend ces valeurs dans l'intervalle [Ø,2] et designe l'un des trois

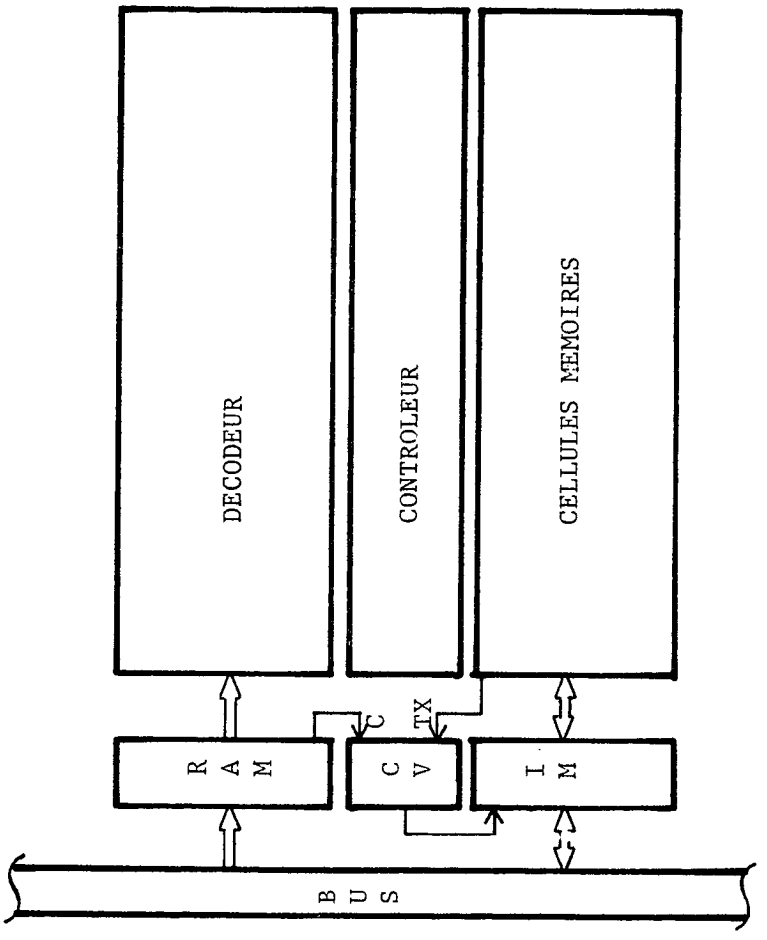


Figure 12: architecture de la memoire

phonemes x;
- C vaut Ø ou 1 selon que y est une voyelle ou bien une Consonne.

La memoire est divisee en 3 blocs de 2Ø mots chacuns. La valeur B sert a designer l'un des 3 Blocs, la valeur A indique l'Adresse d'un mot a l'interieur d'un bloc. Chaque mot contient 2 informations. La premiere, notee Q, represente une valeur de confusion qc(x/y). Afin de reduire la taille de la memoire, Q est code sur 11 bits, 4 bits pour la partie entiere et 7 bits pour la partie fractionnaire, ce qui donne une precision suffisante pour le codage des probabilites de confusion. Lorsqu'une valeur memoire est transferee sur le bus, elle est completee par 5 uns sur les poids forts. La seconde, codee sur 1 bit, note TX, donne le type du phoneme x.

Lorsqu'une lecture memoire est executee avec le triplet (A,B,C), le bit TX du mot lu est compare au bit C. Si TX et C sont egaux, la lecture est valide. Si par contre, TX et C sont differents, la valeur lue est invalide, et doit etre remplacee par la valeur Min.

La memoire est constituee de 5 parties principales: un tableau de cellules memoires (note CM), un registre adresse (note RAM), un decodeur (note DC), un controleur (note CT), une interface memoire (note IM). L'architecture de la memoire est schematisee sur la figure 12.

Les differentes parties constituant la memoire sont detailles dans les paragraphes suivants.

5.2.1-Le tableau de cellules memoires

Les cellules memoires sont organisees en un tableau de 12 lignes par 6Ø colonnes. Chaque colonne represente un mot memoire. Le circuit d'une cellule memoire est represente sur la figure 13.

Cette cellule est une cellule memoire dynamique a 3 transistors (notes T1, T2, T3). La capacite associee a la grille du transistor T2 sert a memoriser une information elementaire. Un seul bus est utilise pour transferer les informations de ou vers la cellule memoire. Le bus est precharge pendant PHI1, les operations de tranfert s'effectuent pendant PHI2. Les commandes de lecture et d'ecriture associees a la cellule, doivent etre des signaux d'horloge synchronises avec PHI2.

L'ecriture d'une valeur dans une cellule est effectuee de la facon suivante. L'information a stocker est presentee sur le bus, et la commande d'ecriture (notee ECR) est activee. Le transistor T1 devient passant et charge la capacite associee a la grille du transistor T2, avec la tension se trouvant sur le bus. Par consequent, la cellule memorise l'information se trouvant sur le bus. A noter que cette methode peut donner naissance a un probleme de partage de charge (charge sharing) entre le bus et la capacite associee au bit memoire. En effet, si l'une seulement des capacites est chargee, l'operation d'ecriture provoque un partage de charge entre les deux capacites. Cependant, le bon fonctionnement du systeme est assure a condition que la capacite associee au bus soit tres superieure a celle de la grille du transistor T2. Nous avons pu verifier que cette condition etait valide, dans la mise en oeuvre qui a ete faite.

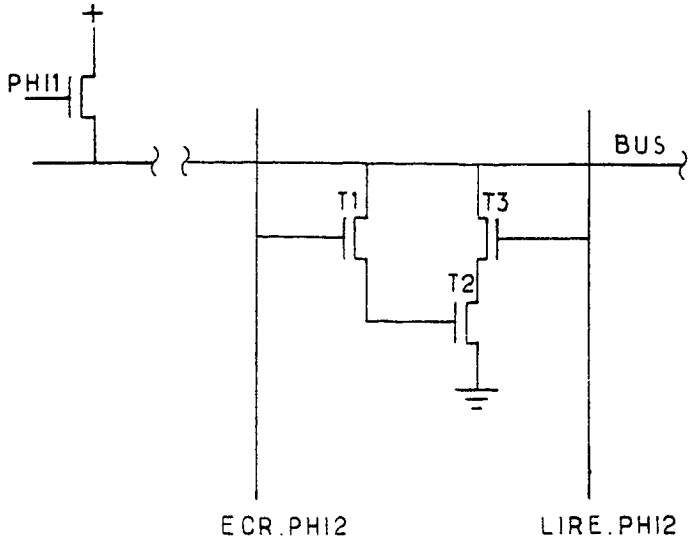


Figure 13: circuit d'une cellule memoire

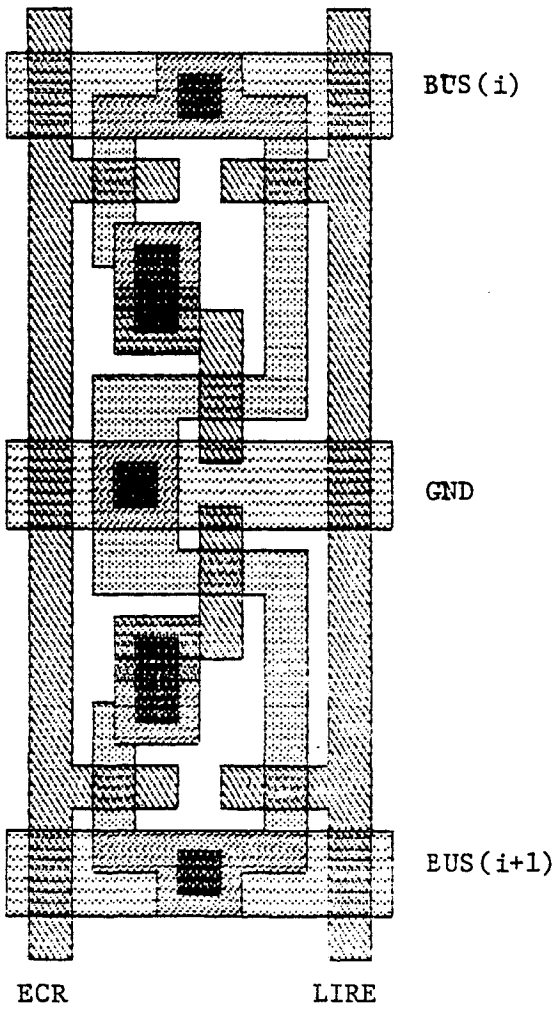


Figure 14: dessin d'une double cellule memoire

La lecture s'opere en activant pendant PHI2, la commande LIRE. Le transistor T3 devient passant. Deux cas sont a considerer. D'une part, si la valeur stockee sur la grille de T2 vaut 1, le bus est alors relie a la masse a travers les transistors T2 et T3. Si la valeur stockee sur la grille de T2 vaut Ø, par contre, le bus reste charge. Par consequent, on constate que la valeur lue est le complement de la valeur ecrite.

La cellule decrite precedemment est une cellule memoire dynamique car elle ne peut stocker une information indefiniment, a cause de courants de fuite au niveau du transistor T2 qui dechargent la capacite de la grille de T2. Un rafraichissement des cellules memoires sera donc necessaire periodiquement (quelques millisecondes). Deux raisons principales ont motive le choix de cellules memoires dynamiques: un encombrement reduit sur le silicium et une consommation d'energie pratiquement nulle.

Mise en oeuvre

Le dessin a ete fait de telle sorte que deux cellules voisines puissent partager la meme masse. La figure 14 donne le dessin des masques d'une double cellule memoire. Deux cellules sont placees l'une sur l'autre. La seconde cellule etant symetrique a la premiere par rapport a la masse.

5.2.2-Le registre adresse

Le registre adresse est utilise pour memoriser l'adresse a laquelle la prochaine operation memoire doit etre effectuee. Il est constitue de 8 bits dont la signification est la suivante:

- bits Ø-1: codage de B, adresse de Bloc;
- bits 2-6: codage de A, Adresse dans le bloc;
- bit 7 : codage de C, type de phoneme.

Le schema d'un bit du registre est donne sur la figure 15. La valeur provenant du bus est tout d'abord inversee (comme le preconise le fonctionnement du bus), puis charge dans un registre dynamique dont la commande de chargement (note CH.PHI1) est synchronisee sur PHI1.

L'utilisation d'un registre dynamique se justifie tres bien puisque le registre adresse memorise l'adresse de la prochaine operation memoire. Par consequent, l'adresse ne doit etre memorisee que pendant quelques cycles d'horloge. Le bit d'information memorise dans le registre est amplifie et fourni, sous ses formes normale et complementee, au decodeur.

Mise en oeuvre

Les cellules registres sont empilees les unes sur les autres. Des cellules amplificatrices appelees ADRSPB (amplification sans inversion) et ADRISP (amplification avec inversion) sont egalement placees les unes sur les autres. Cependant, une fois sur deux, les cellules ADRSPB et ADRISP sont placees tete en bas. La figure 16 montre un empilement de 3 cellules de registres avec les amplificateurs associes.

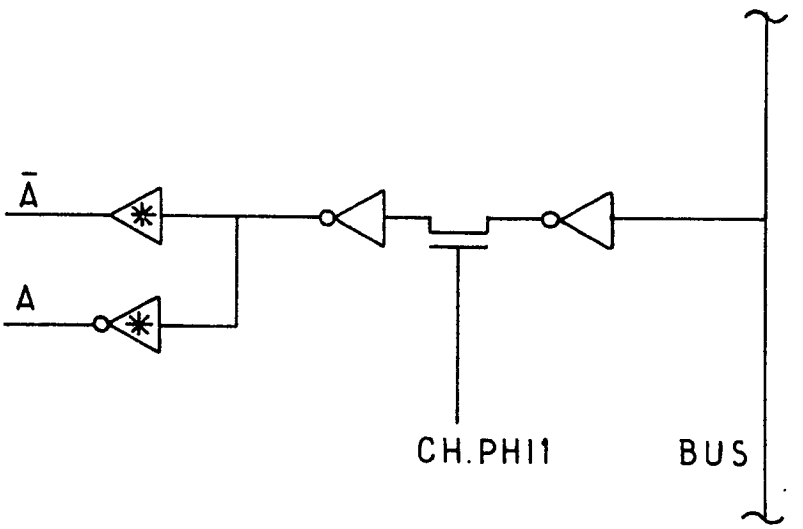


Figure 15: cellule de registre adresse memoire

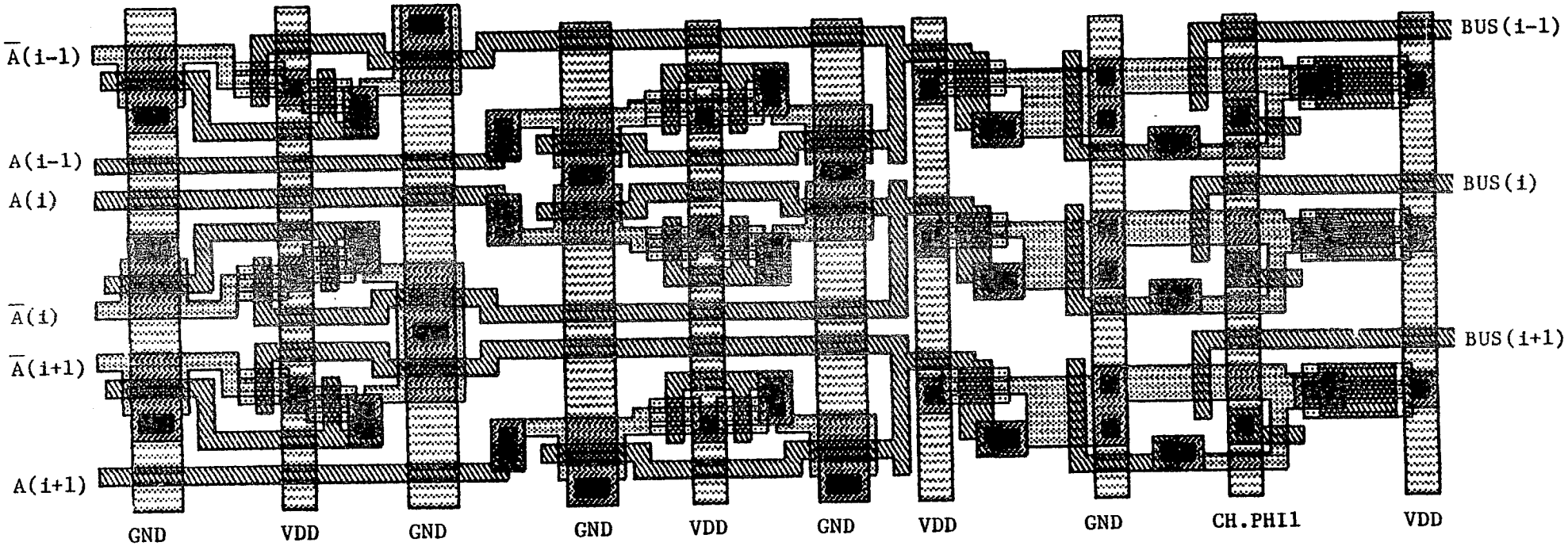


Figure 16: dessin de 3 cellules du registre adresse memoire

5.2.3-Le decodeur

Lorsqu'une commande d'acces memoire a ete emise, le role du decodeur est d'activer la commande LIRE ou ECR associee aux cellules memoires du mot adresse. Le decodeur est represente sur la figure 17. Le decodeur recoit 2 types de signaux d'entree:
- des signaux d'adresse;
- des signaux de lecture et d'ecriture (LIRE et ECR).
et fournit, en sortie, 2 signaux Lire(i) et Ecr(i) pour chaque mot memoire i. Chaque signal d'entree provenant du registre adresse est amplifie sous ses 2 formes: normale et complementee. Chaque signal de sortie est engendre par un circuit NOR a entrees multiples.

5.2.4-Le controleur

Le role du controleur est double. Il doit, d'une part, memoriser les commandes de lecture et d'ecriture provenant du decodeur pendant la phase PHI2, puis, d'autre part, emettre des signaux de commande amplifies pendant PHI1. La figure 18 represente le circuit du controleur. Pendant PHI2, la commande LIRE ou ECR provenant du decodeur est memorisee sur la grille du transistor T1. Pendant PHI1, la valeur de commande est amplifiee et envoyee vers les cellules memoires.

5.2.5-L'interface memoire

Ce module sert d'interface entre le bus memoire (note BM) et le bus general (BG). Lorsqu'une operation d'ecriture memoire est effectuee, l'information a ecrire, en provenance de BG est tout d'abord stockee dans un registre appele REC avant d'etre transferee sur BM pour une ecriture effective. Lorsqu'une operation de lecture est demandee, un processus similaire se deroule. La donnee provenant de BM est memorisee dans un registre appele RLE, apres avoir subi certaines transformations, puis est envoyee sur BG. Ces transformations sont de deux ordres. D'une part, du a la structure d'une cellule memoire a 3 transistors, une valeur lue doit etre complementee. D'autre part, l'organisation de la memoire, decrite plus haut dans la paragraphe 5.2, necessite un controle de validite d'un mot lu. En particulier, dans le cas d'un mot invalide, la valeur Ø doit etre rangee dans le registre RLE.

Le circuit represente sur la figure 19a engendre un signal note DV (Donnee Valide), qui vaut Ø lorsque le mot lu est valide et reciproquement vaut 1 lorsqu'il est invalide. DV est determine a partir de trois signaux d'entree: TXbar et Cbar decrits precedemment, et un signal R (pour Rafraichissement). La commande R est utilisee pour preciser qu'un controle de validite d'une donnee lue en memoire est necessaire. En effet, il n'est pas toujours necessaire d'effectuer de controle. La memoire etant dynamique, elle doit etre rafraichie de temps en temps. Dans ce cas, la valeur lue en memoire (y compris le bit TX) est reecrite sans aucune modification. La figure 19c donne de facon plus detaillee le circuit qui a ete effectivement mis en oeuvre.

La figure 19b donne la table de verite associee au circuit. Lorsque R vaut 1, cas ou aucun controle de validite n'est demande, DV vaut Ø (les donnees sont valides) quelles que soient les valeurs de

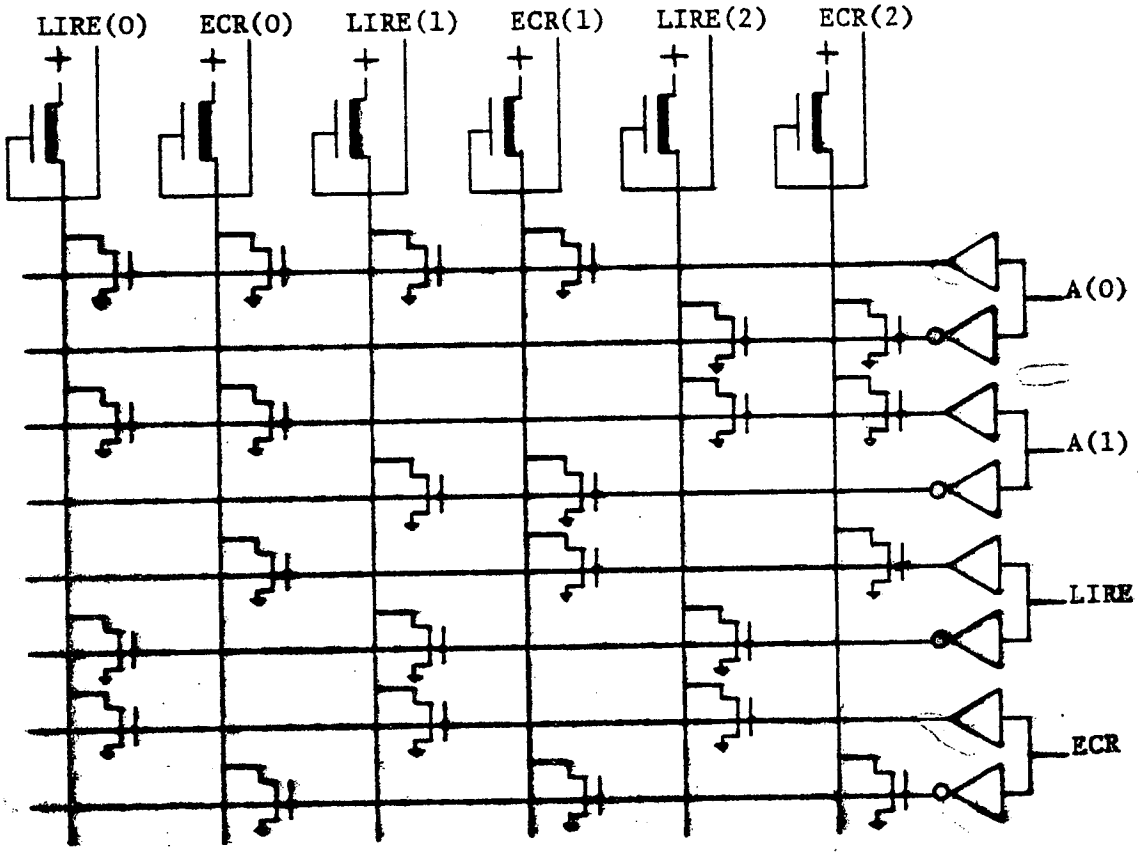


Figure 17: decodeur memoire

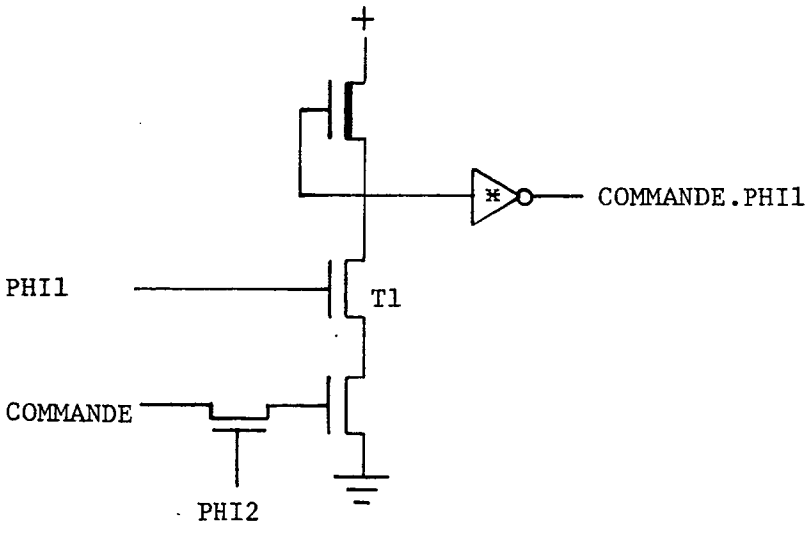


Figure 18: circuit de controle d'operations memoire

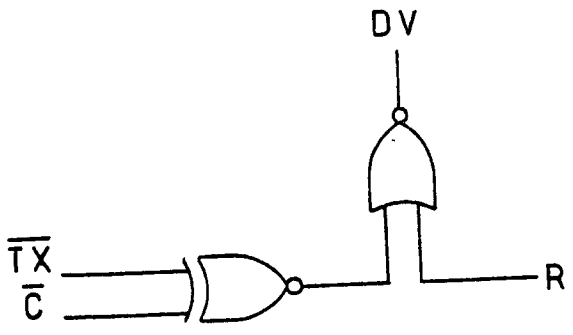


Figure 19a: circuit de controle de validite

$\overline{TX}$	$\overline{C}$	R	DV
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Figure 19b: table de verite du controle de validite

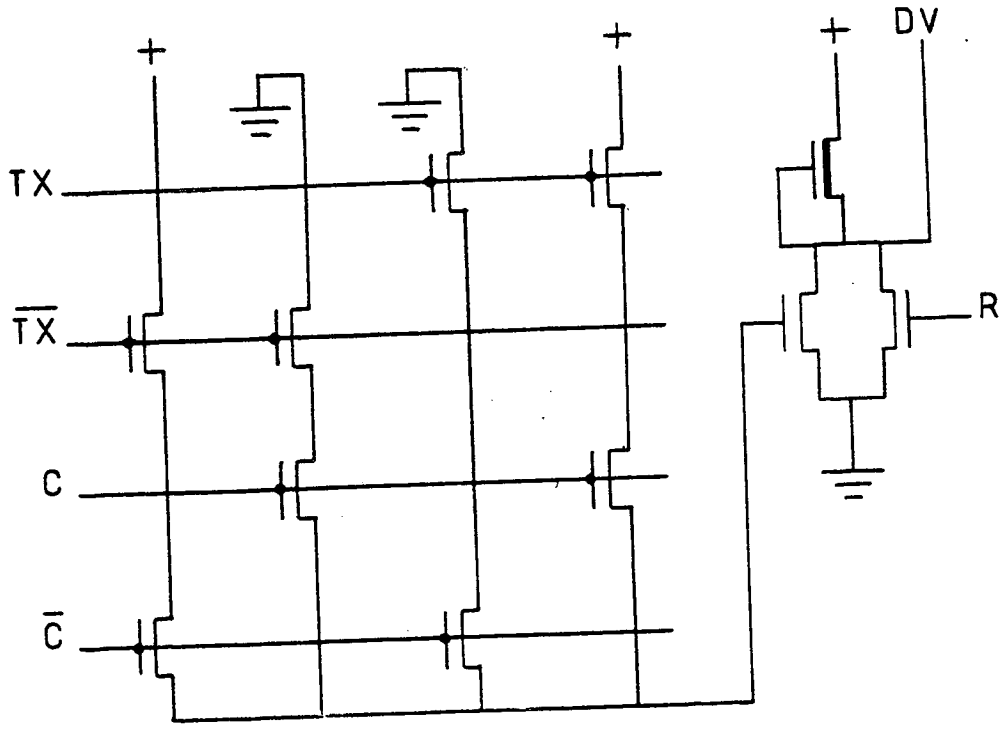


Figure 19c: mise en oeuvre du circuit de controle

TXbar et Cbar. Lorsque R vaut 1, par contre, les donnees ne sont valides (DV=Ø) que si TXbar et Cbar sont egaux.

Le circuit associe aux registres REC et RLE est represente sur la figure 2Ø. Le registre REC est represente sur la figure 2Øa. La donnee provenant du bus BG est tout d'abord inversee (a cause du principe de fonctionnement du bus), puis memorisee dans un registre dynamique grace a la commande CH.PHI1. La commande ECR.PHI2 permet de tranferer sur le bus BM, la donnee memorisee (qui a subi 2 inversions successives).

Le registre RLE (figure 2Øb) est similaire a REC. La commande LIRE.PHI2 permet la memorisation de la donnee lue. La donnee enregistree est envoyee sur le bus BG grace a la commande EMET.PHI1.

Sur la figure 2Øc, est represente le circuit d'interface entre le bus BM et le registre RLE. La table de verite associee a ce circuit (figure 2Ød) indique d'une part que si la donnee est valide (DV=Ø), la donnee lue en memoire (DL) est complementee (voir fonctionnement d'une cellule memoire), et que, d'autre part, si DV=1 la valeur Ø est fournie en entree du registre RLE.

Cas particulier

Le module IM est compose de 12 cellules elementaires decrites precedemment. Cependant la cellule associee au bit TX est legerement differente des autres. En effet son fonctionnement est le suivant:
- lorsqu'un rafraichissement est demande (R=1), la valeur lue en memoire est envoyee sur le bus BG (apres complementation);
- lorsque le rafraichissement n'est pas demande, la valeur 1 est envoyee sur le bus BG, quelle que soit la valeur lue.

Le circuit correspondant a la cellule TX est represente sur la figure 21. L'interface entre le bus BM et le registre RLE differe de celle decrite figure 2Øc. La table de verite correspondant a ce circuit est donnee sur la figure 21d: si R=1 (rafraichissement), la donnee lue (DL) est complementee; par contre si R=Ø, la valeur 1 est engendree.

Mise en oeuvre

Les cellules de IM sont empilees les une sur les autres permettant ainsi aux commandes associees de se propager verticalement. La figure 22 represente le dessin de l'empilement de 2 cellules dont l'une est la cellule TX.

5.2.5-Remarque

La memoire contenant des mots de 12 bits, 4 bits supplementaires doivent etre fournis pour completer a 16 bits. Les donnees en memoire sont toujours negatives, donc 4 bits 1 sont envoyes sur les poids forts du bus.

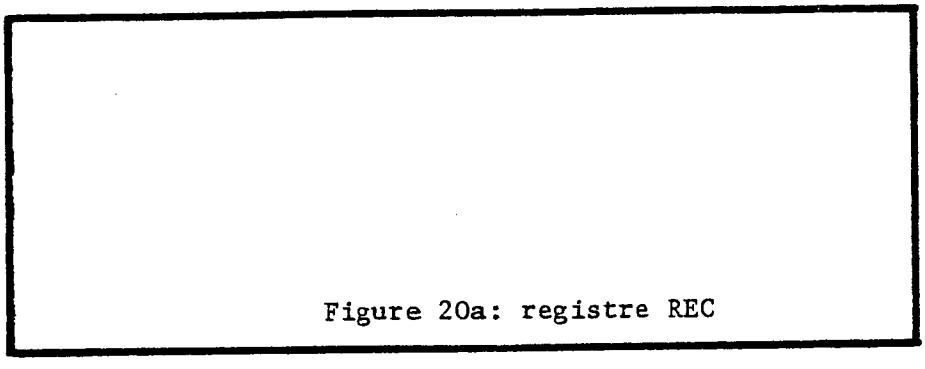


Figure 20a: registre REC

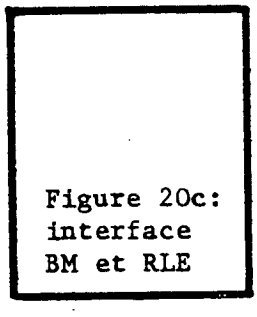


Figure 20c: interface BM et RLE

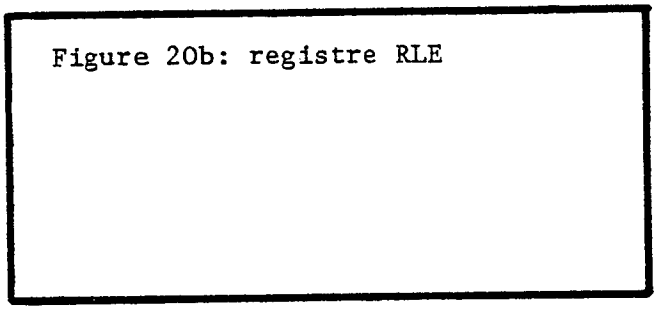
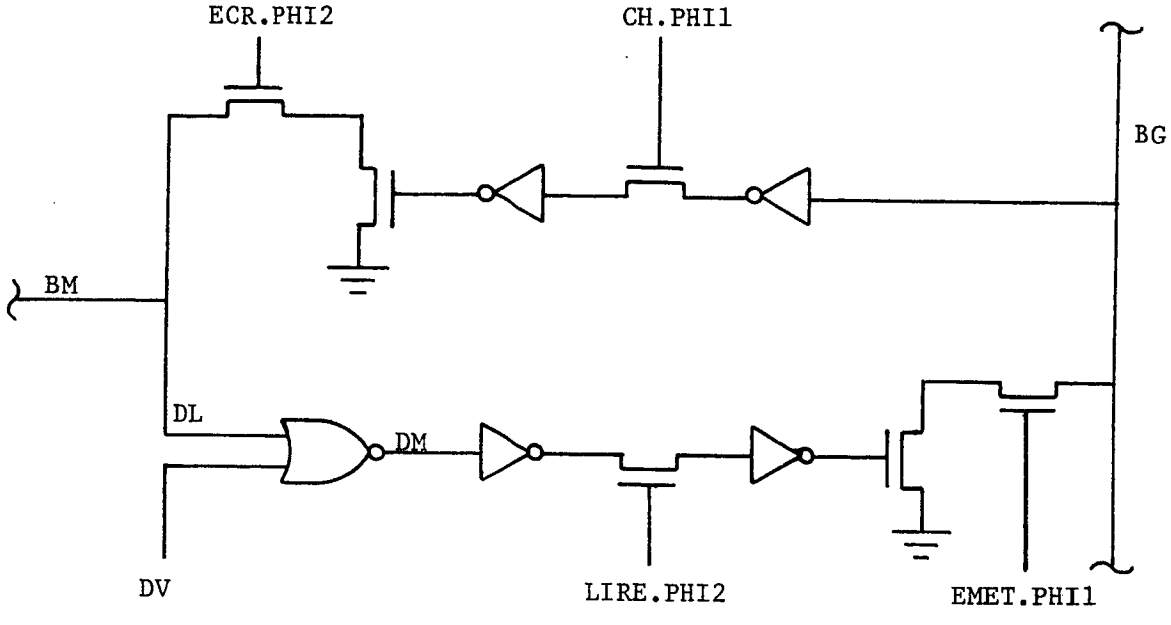


Figure 20b: registre RLE



DV	DL	DM
0	0	1
0	1	0
1	0	0
1	1	0

Figure 20d: table de verite associee au circuit interfacant le bus BM et le registre RLE

Figure 20: interface memoire

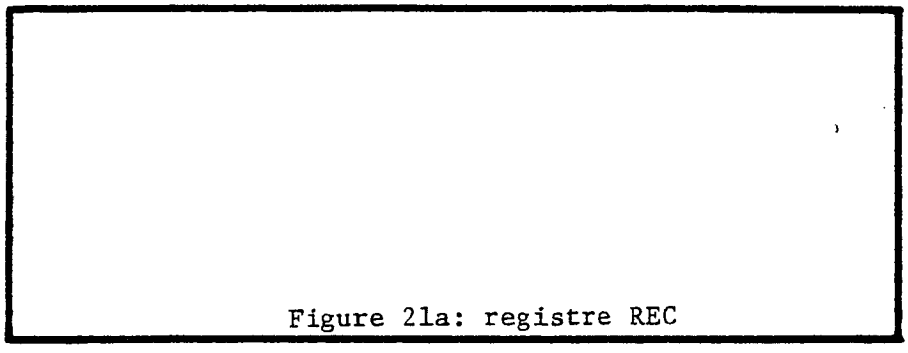


Figure 21a: registre REC

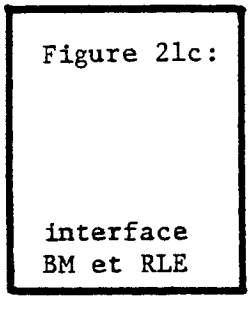


Figure 21c:

interface
BM et RLE

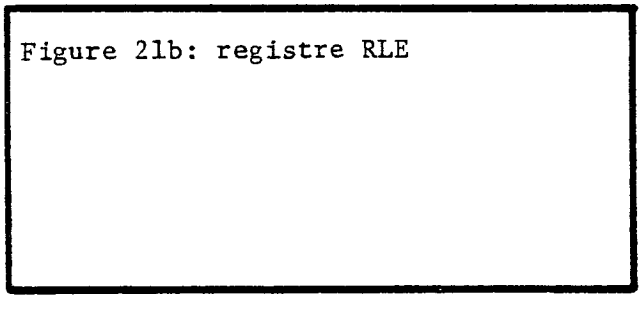
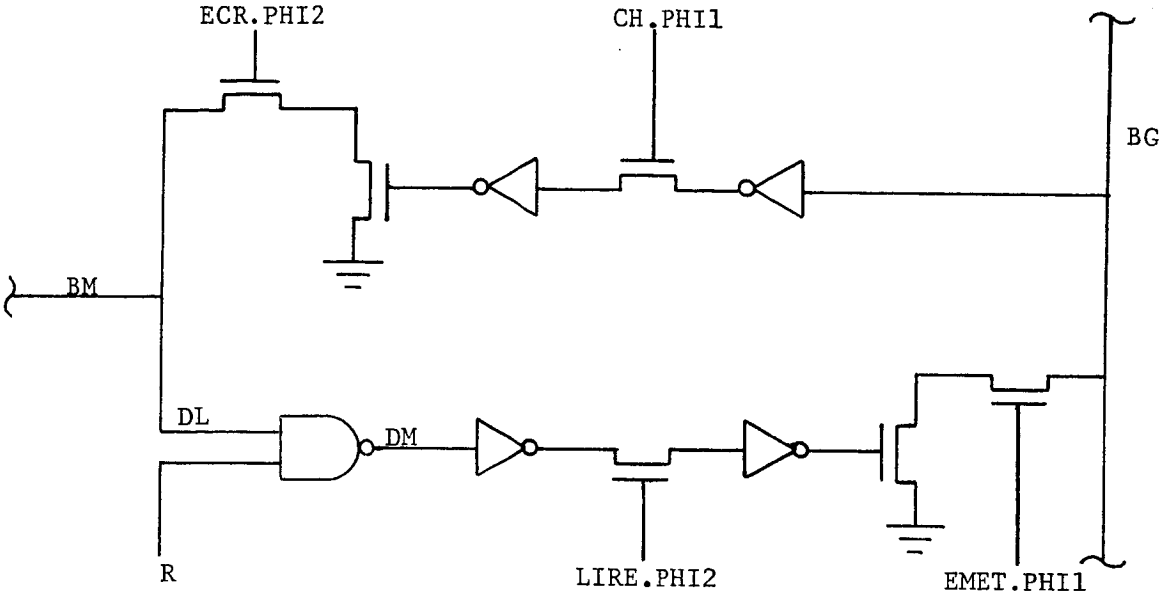


Figure 21b: registre RLE



DL	R	DM
0	0	1
0	1	1
1	0	1
1	1	0

Figure 21d: table de verite du circuit interfacant le bus BM et le registre RLE

Figure 21: cellule associee au bit TX

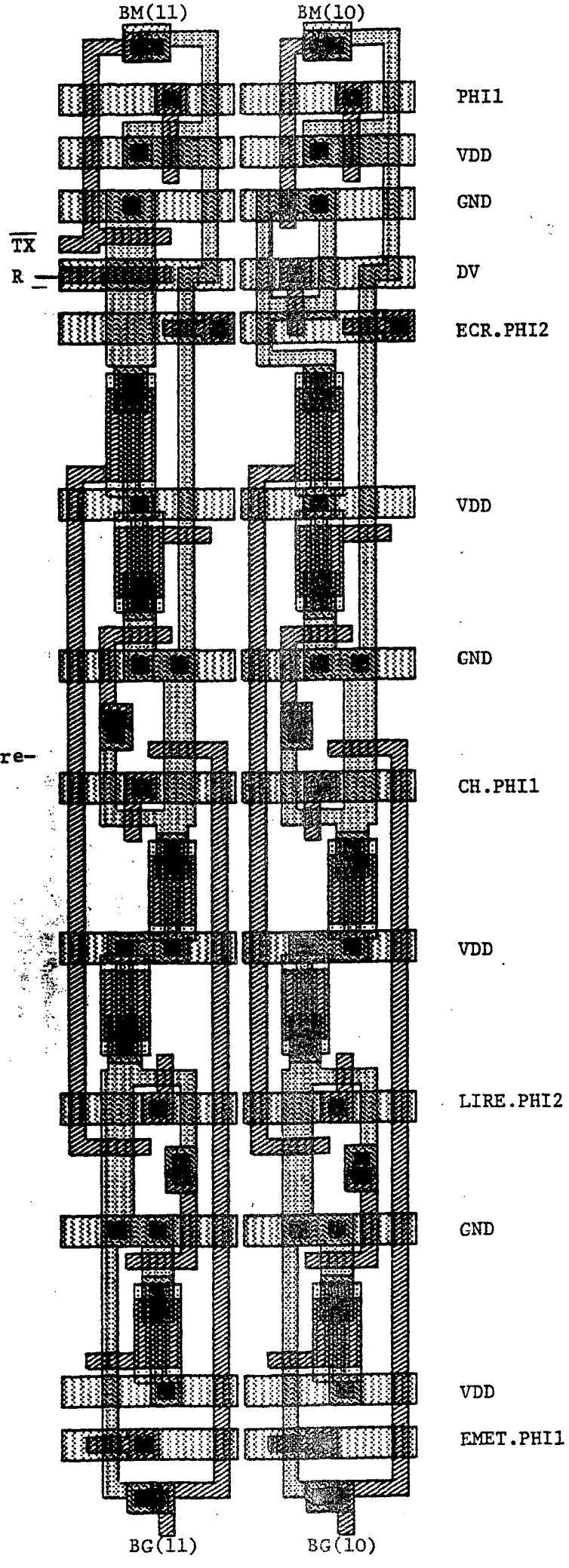


Figure 22: dessin de 2 cellules de IM. Le dessin de gauche represente la cellule TX

5.3-Le module Z

Une maniere generale de mettre en oeuvre une fonction logique combinatoire ayant n entrees et m sorties est d'utiliser une memoire de 2**n elements de m bits chacuns. Les n entrees sont utilisees pour adresser la memoire, et le mot accede fournit les m valeurs de sorties. Une memoire a acces en lecture seulement, appelee memoire morte, est generalement utilisee. Le probleme majeur associe a cette technique est du a la taille de la memoire ainsi obtenue. En effet cette methode revient a coder, exhaustivement, la table de verite de la fonction F, sans se soucier de simplifications possibles.

Le module Z sert a tabuler la fonction Z definie dans le paragraphe 4 de la facon suivante:

(13) $Z: t \dashrightarrow Z(t)=Log(1+2^t)$

Pour mettre en oeuvre cette fonction, la methode la plus simple consiste a tabuler cette fonction a l'aide d'une memoire morte. Cependant les valeurs de Log(u) etant codees sur 16 bits, une memoire morte de 64K mots est necessaire. Vue la taille importante de cette memoire, cette solution a ete rejetee. Comme nous allons le montrer plus loin, l'utilisation d'un PLA (Programmable Logic Array) pour tabuler Z est une solution interessante.

Ce paragraphe se compose de 3 parties. Une breve definition d'un PLA est donnee dans une premiere partie. On montre, dans une seconde partie, comment un PLA peut etre utilise pour tabuler la fonction Z. La troisieme partie concerne la mise en oeuvre complete du module Z.

5.3.1-Definition d'un PLA

Notons Bp l'ensemble des vecteurs de p composantes booleennes. Soit F une fonction booleenne definie de Bn vers Bm. Soit Fi, (1<=i<=m) la composante de F, definie de Bn vers B, telle que si Y appartient a Bm, X appartient a Bn et Y=F(X) alors Fi(X)=Y(i). Chaque fonction Fi, peut s'exprimer d'une facon simplifiee, sous forme d'une somme de monomes. Soit M l'ensemble des monomes necessaires a la definition de toutes les fonctions Fi, la technique utilisee dans un PLA consiste a decoder les seuls monomes appartenant a M. Le PLA est constitue de 2 parties appelees matrice ET et matrice OU. La matrice ET est formee de 2xn colonnes et Card(M) lignes. Les colonnes representent les entrees sous leurs 2 formes: forme normale et forme complementee. Chaque ligne de la matrice ET decode un monome de M, en ce sens qu'elle effectue le ET logique (d'ou le nom de la matrice) des termes constituants le monomes. La matrice OU est formee de m colonnes et Card(M) lignes. Chaque ligne represente un monome de M. Chaque colonne est associee a une fonction Fi, puisqu'elle effectue le OU logique des monomes constituants Fi.

5.3.2-Utilisation d'un PLA pour tabuler Z

L'idee d'utiliser un PLA pour tabuler Z est nee de la remarque suivante. Soit u une valeur definie dans l'intervalle]Ø,1]. Il est clair que Log(u) prends ses valeurs dans l'intervalle]-inf,Ø] et

Log(1+u) dans l'intervalle]Ø,1]. Si l'on considere que les nombres sont codes de la maniere decrite dans le paragraphe 4.1.3 (codage sur 16 bits, dont les 9 bits de poids fort representent la partie entiere et les 7 bits de poids faible, la partie fractionnaire), on constate que l'intervalle]Ø,1] est codable sur 7 bits et ne contient que 128 valeurs (on notera E cet ensemble de valeurs). Par contre, l'intervalle]-inf,Ø] necessite un codage sur au moins 15 bits, et contient 32K valeurs (soit S cet ensemble).

Soit Z' la fonction approchee de Z, definie de E vers S. Les remarques precedentes montrent que Z' est surjective. Par ailleurs, la fonction Z' etant monotone croissante, on en deduit que les valeurs de l'espace de depart donnant la meme valeur dans l'espace d'arrivee, sont consecutives. Soient v(i,1),...,v(i,k(i)), 1<=k(i)<=n, les valeurs de Bn qui produisent la valeur V(i) de Bm par application de Z', la remarque precedente indique que les valeurs v(i,1),...,v(i,k(i)) ne different que par leurs bits de poids faibles. L'utilisation d'un PLA suggere le regroupement des monomes M(i,1),...,M(i,k(i)) associes a v(i,1),...,v(i,k(i)) en un seul monome M(i) tel que:

$$M(i)=\sum_{j=1}^{k(i)} M(i,j) \qquad \text{(sommation booleenne)}$$

Malheureusement, dans la plupart des cas, cette condition n'est pas realisable. Une solution a ce probleme consiste a approcher Z' par une fonction Z'' qui permet de realiser la condition precedente.

Soit M'(i) un monome qui se decompose de maniere canonique, sous forme d'une somme de monomes M'(i,j) de la facon suivante:

$$M'(i)=\sum_{j=1}^{k'(i)} M'(i,j)$$

Posons v'(i,j), la valeur associee a M'(i,j). La resolution du probleme consiste a trouver un ensemble de monomes, note M, dont les elements sont notes M'(i), qui satisfait les conditions suivantes:

(C1) (∀i,∀j, i#j) M'(i)+M'(j)#M'(i);
et M'(i)+M'(j)#M'(j)

(C2) (∀V(i), V(i) appartenant a S),
il existe v'(i,j) associe a M'(i,j) de M'(i)
tq Z'(v'(i,j))=V(i)

(C3) (∀v, v appartenant a E),
il existe M'(i) englobant le monome Mv associe a v

(C4) il existe une bijection entre M et S

La condition (C1) exprime que les monomes de M sont disjoints, c'est a dire qu'ils couvrent des valeurs differentes. La condition (C2) exprime que toute valeur de l'espace d'arrivee S de la fonction Z' est representee dans un monome. Par symetrie, la condition (C3) necessite que tout monome associe a une valeur de l'espace de depart

soit couvert par un monome M'(i). Enfin, la condition (C4) evite qu'un meme monome puisse etre utilise pour deux valeurs de l'ensemble d'arrivee. On peut montrer que plusieurs ensembles M satisfont les conditions precedentes. La fonction Z'' cherchee, approchee de la fonction Z', est la fonction definie de E vers S, telle qu'a tout element v'(i,j) de E associe a M'(i,j) on affecte la valeur V(i) associee a M'(i), au sens de la condition (C2).

L'algorithme suivant a ete utilise pour determiner un ensemble M. Posons R la fonction reciproque de Z et R' la fonction approchee de R definie de S vers E. La fonction Z' etant surjective, il est clair qu'il existe des elements de E n'ayant pas d'antecedant. Posons E', le sous-ensemble de E, tel que tout element de E' possede un antecedant par R'. Soient a(i-1), a(i), a(i+1) trois elements consecutifs de E'. L'algorithme permettant de determiner les monomes M'(i) est le suivant:

```
pour tout triplet (a(i-1),a(i),a(i+1))
faire
   k:=a(i-1); l:=a(i); m:=a(i+1);
   tant que (k/2 # l/2) et (l/2 # m/2)
   faire
      k:=k/2; l:=l/2; m:=m/2
   fait;
   M(i):= monome associe a l
fait
```

Soit m(i) le monome associe a a(i). Le principe de l'algorithme est de determiner le plus grand monome M(i) englobant m(i) verifiant la condition (C1). La methode utilisee consiste a supprimer successivement les bits de poids faible (division par 2) des valeurs a(i-1), a(i) et a(i+1). Les monomes associes aux nouvelles valeurs englobent ainsi les monomes associes aux precedentes valeurs. Le traitement se poursuit tant que les valeurs ainsi obtenues sont differentes. En effet, si deux valeurs deviennent egales, cela signifie que les monomes associes sont egaux et donc que la condition (C1) n'est pas verifiee. On montre que le monome obtenu M(i) verifie les conditions (C1), (C2) et (C4). La condition (C1) est verifiee par construction meme de l'algorithme. M(i) verifie la condition (C2), puisque par construction, a(i) appartient a E'. Il est clair que la condition (C4) est egalement realisee par construction, puisque pour chaque element de E',l'algorithme determine un monome. La condition (C3), par contre, n'a pu etre verifiee que par l'experience.

5.3.3-Mise en oeuvre du module Z

Le module Z est forme de 3 parties comme le montre la figure 23: un registre d'entree note RPLA, un PLA et un controleur de sortie.

5.3.3.1-Le registre d'entree

RPLA est un registre de 11 bits. En effet, la table de verite du PLA produite par l'algorithme decrit dans le paragraphe 5.3.2, montre que les bits 10 a 14, n'interviennent que lorsqu'ils valent tous 1. Il est donc interessant de remplacer ces 5 bits par leur ET logique. Par ailleurs, la table de verite montre que le bit 0 n'est jamais utilise. Les 11 bits du RPLA ont la signification suivante:

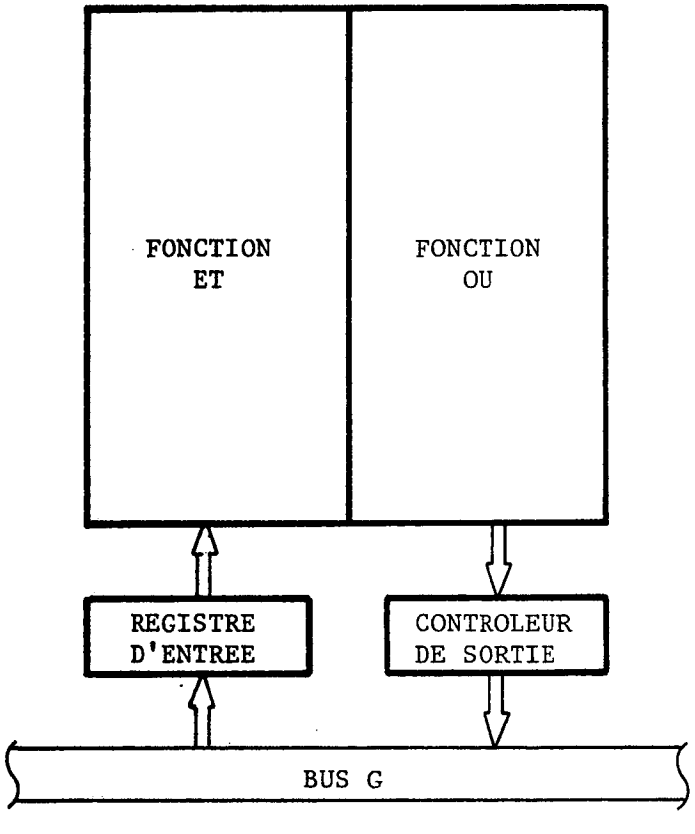


Figure 23: architecture du module Z

- les bits Ø a 8 sont connectes aux bits 1 a 9 du bus;
- le bit 9 est connecte au bit 15 du bus (bit de signe)
- le bit 1Ø represente le ET logique des bits 1Ø a 14 du bus.

Pour les bits Ø a 9, le registre d'entree a la structure representee sur la figure 24.

La figure 25 represente le circuit associe au bit 1Ø du PLA. Le NOR a 5 entrees est utilise pour calculer la fonction suivante:

$$B14.B13.B12.B11.B1Ø = \overline{B14+B13+B12+B11+B1Ø}$$

5.3.3.2-Le controleur de sortie

Le PLA fournit des valeurs positives entre Ø et 1 et donc 8 bits sont necessaires pour leur codage: 1 bit avant la virgule et 7 bits apres. Lorsque le resultat du PLA est envoye sur le bus, les 8 bits de sortie sont envoyes sur les poids faible du bus, et 8 valeurs Ø sont envoyees sur les poids forts. La figure 26 montre le circuit associe a une cellule du controleur de sortie.

5.3.3.3-Le PLA

Le PLA est forme de 11 entrees, 8 sorties et 128 termes produits. Il a ete engendre automatiquement par un programme appele MKPLA [5]. La mise en oeuvre de ce PLA est decrite en detail dans Mead et Conway [6] et Hon et Sequin [7].

5.4-L'unite arithmetique et logique (UAL)

Elle est destinee a effectuer les operations necessitees par les differents programmes. Ces operations sont au nombre de 6 et necessitent 1, 2 ou pas d'operandes. Certaines operations positionnent un indicateur de condition (note COND); d'autres operations s'effectuent conditionnellement a la valeur contenue dans cet indicateur. Si l'on note A et B les operandes et R le resultat de l'operation, les operations de l'UAL sont les suivantes:

- addition	A + B --> R	
- soustraction	A - B --> R	
- complementation conditionnelle	* B --> R	vaut - B si COND B sinon
- incrementation	B + 1 --> R	
- soustraction conditionnelle	A * B --> R	vaut A-B si COND A sinon
- zero	Ø --> R	

L'unite arithmetique et logique utilisee est celle decrite dans Mead et Conway [6].Cette UAL est formee principalement de trois modules:
- un module note KILL charge d'arreter la propagation de la retenue;
- un module note PROP qui propage la retenue;
- un module, note RESU, dont le but est de fournir le resultat d'une operation.

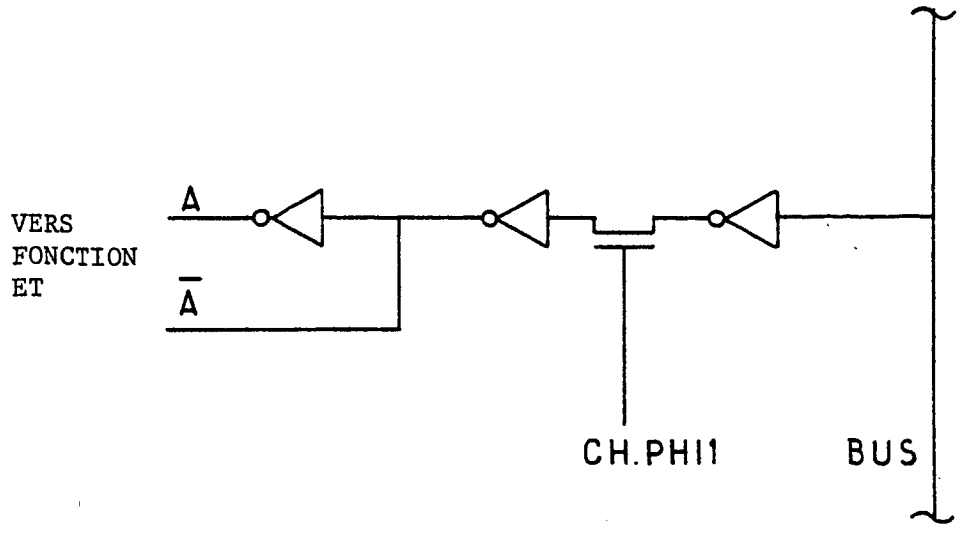


Figure 24: registre entree PLA

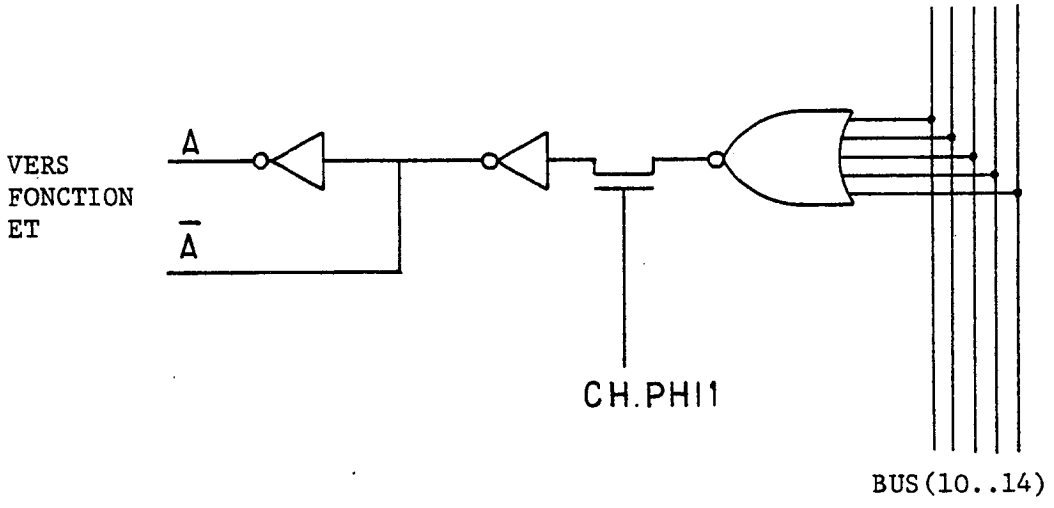


Figure 25: bit 10 du registre entree PLA

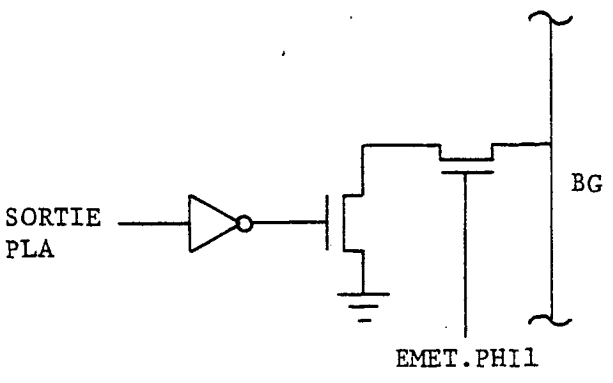


Figure 26: cellule de sortie PLA

Les modules KILL, PROP, RESU sont controles par trois groupes de signaux de 4 bits chacun, notes respectivement K, P, R. Par ailleurs, l'UAL recoit un signal, note Cin, de retenue entrante. Cette UAL est tres generale puisqu'a chaque combinaison de K, P, R et Cin est associee une operation arithmetique ou logique (exemples: addition, soustraction, incrementation, et logique, ou logique, decalage). Le tableau suivant donne les valeurs de K, P, R, Cin pour effectuer les operations decrites precedemment en fonction de la valeur de la condition.

Operation a effectuer	Condition	K	P	R	Cin	Operation effectuee
A+B	-	1	6	6	Ø	A+B
A-B	-	2	9	6	1	A-B
*B	1	1Ø	5	6	1	-B
*B	Ø	Ø	1Ø	12	Ø	B
B+1	-	5	1Ø	6	1	B+1
A*B	1	2	9	6	1	A-B
A*B	Ø	Ø	12	12	Ø	A
Ø	-	Ø	Ø	Ø	Ø	Ø

Trois registres sont attaches a l'UAL comme le decrit la figure 27a. Les registres RA et RB sont utilises pour memoriser provisoirement les 2 operandes. Le registre RR memorise le resultat de l'operation. Les registres RA et RB sont charges pendant PHI1, puis l'operation est effectuee sur l'UAL et le resultat charge dans RR pendant PHI2. La figure 27b donne les circuits des registres RA, RB et RR. On remarque que le registre RA est charge a partir du bus alors que RB ne peut etre charge qu'avec la valeur de RR.

L'indicateur de condition est memorise dans un registre dynamique de 1 bit. Il est charge avec le bit de poids fort du registre RR. Ce bit correspond, en fait, au signe du resultat de l'operation effectuee. Toutes les operations ne chargent pas l'indicateur COND. Les seules operations qui le positionnent sont les suivantes: A+B, A-B, B+1.

5.5-Le tableau de registres

Le programme execute par un processeur (cf programme 2 dans le paragraphe 4.1.2), indique que certaines informations doivent etre memorisees, dans des variables. Ces variables sont de deux types: d'une part, des variables contenant des informations propres a l'algorithme de detection de mots, et d'autre part, des variables de travail qui permettent de memoriser des resultats intermediaires. Les mnemoniques des 16 registres sont les suivants: L, Li, Lc, Lo, YI, YC, YO, X1, X2, X3, I1, I2, I3, NY, Lc_en_attente, T1. Toutes ces variables ont ete regroupees dans un tableau de 16 registres de 16 bits.

Le circuit d'une cellule d'un registre est represente sur la figure 28. Le principe de fonctionnement de ce circuit est d'une part d'effectuer des operations memoires (lecture ou ecriture) pendant PHI1, et, d'autre part, de proceder a un rafraichissement de l'information stockee, pendant PHI2. Une operation d'ecriture, consiste a stocker l'information du bus sur l'entree d'un inverseur. Le second inverseur corrige l'information en sortie du premier inverseur et envoie cette

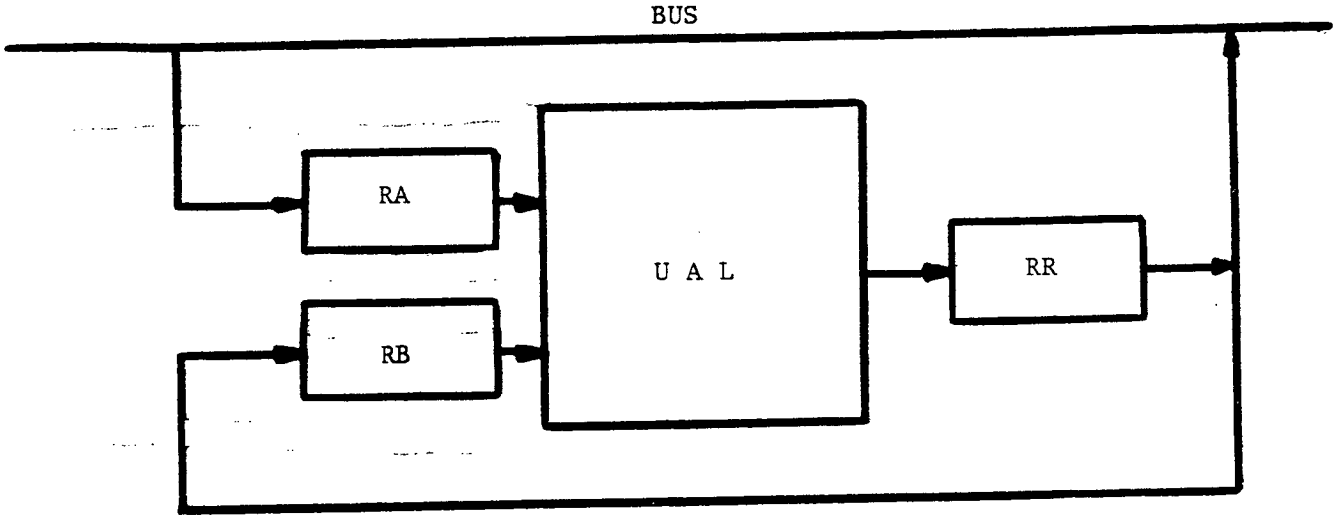


Figure 27a: l'UAL et ses registres

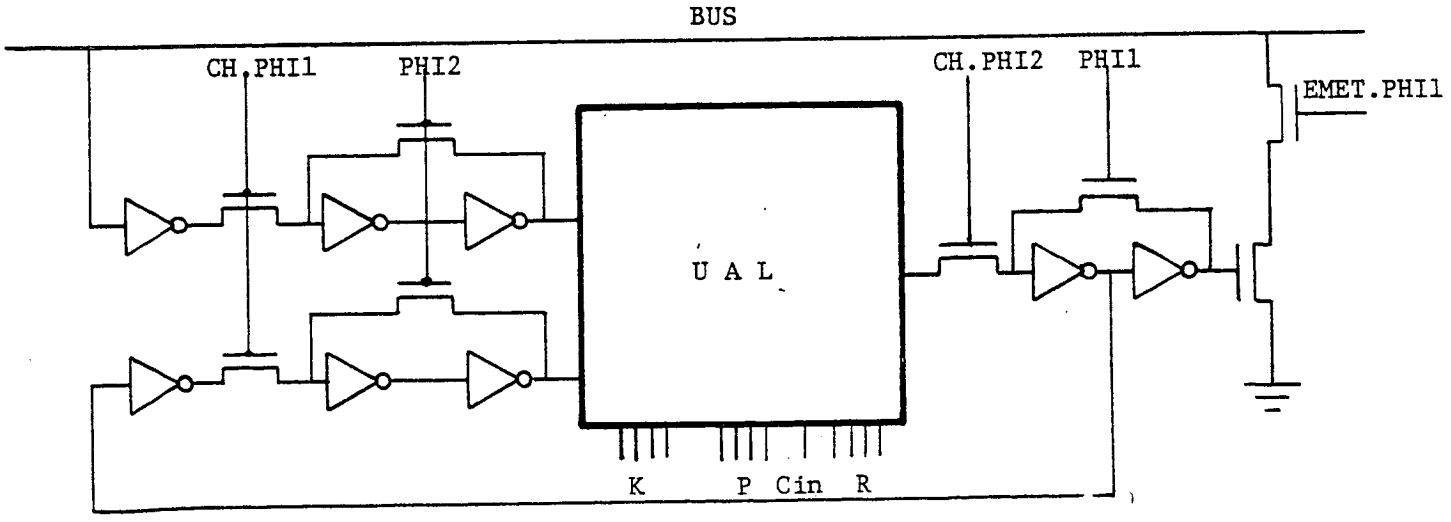


Figure 27b: description des registres de l'UAL

information sur le bus lorsqu'une lecture est effectuee. L'entree du premier inverseur et la sortie du second sont reliees a travers un transistor actif pendant PHI2, permettant ainsi de rafraichir l'information stockee.

On note que les registres ne sont pas connectes au bus BG suivant le principe decrit dans le paragraphe 5.1. En effet, l'information provenant du bus est utilisee directement, sans avoir ete complementee. Reciproquement, l'information stockee dans un registre est envoyee directement sur BG. Chaque cellule registre etant formee principalement de 2 inverseurs, adopter le principe decrit precedemment aurait necessite l'introduction de 2 inverseurs supplementaires, ce qui, pour des raisons d'encombrement, n'etait pas tres realiste. Ainsi les cellules registres recevant directement la valeur du bus, contiennent une information complementee. Lors d'une lecture, la valeur du registre est envoyee directement sur le bus. Par consequent, le bus transfert une valeur complementee comme il se doit.

Les commandes de lecture et d'ecriture, synchronisees avec PHI1, sont engendrees par un circuit de meme type que celui represente sur la figure 18. Pendant PHI2, la commande d'operation memoire est stockee sur la grille du transistor T1; le transistor T2 etant "ouvert", la valeur Ø est fournie en sortie du circuit. Lorsque PHI1 devient actif, l'information precedemment stockee sur T1 est alors disponible sur la sortie.

Mise en oeuvre

Le dessin d'une cellule de registre est represente sur la figure 29. Les cellules d'un meme registre, sont placees les unes sur les autres. Cependant une cellule sur deux est placee apres avoir subi une operation de symetrie sur y. Ceci permet le partage de VDD et GND entre deux cellules voisines.

5.6-Controle des operations internes au processeur

Dans les paragraphes precedents, les differents modules constituant la partie operative du processeur ont ete etudies. A chaque module est associe un certain nombre de commandes necessaires a son fonctionnement. Ces commandes pourraient provenir directement du controleur CG. Cette solution, qui a l'avantage de permettre l'execution de plusieurs commandes en parallele, n'a pas ete retenue a cause du trop grand nombre de commandes qui sont a fournir au chip. La solution adoptee a ete de grouper les commandes et de les coder dans une microinstruction.

Par consequent, cette microinstruction doit etre decodee a l'interieur du processeur. Ce paragraphe est compose de deux parties: le codage des differentes commandes est expose dans un premier paragraphe, les modules effectuant le decodage sont ensuite etudies dans un second paragraphe.

5.6.1-Codage des commandes

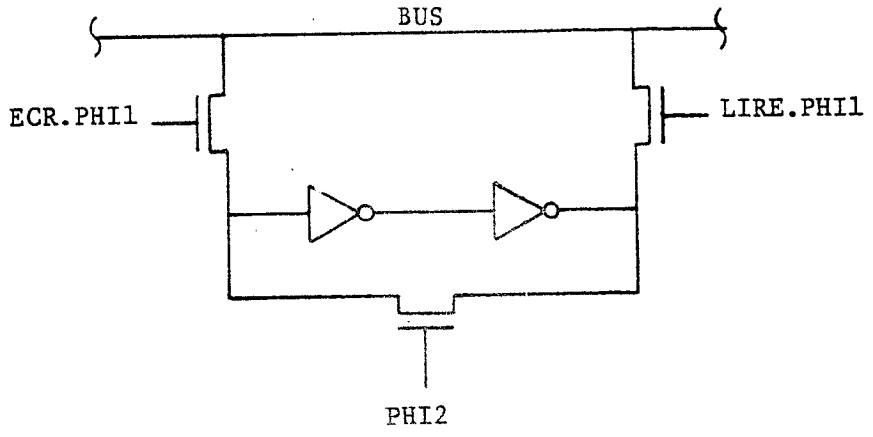


Figure 28: une cellule d'un registre general

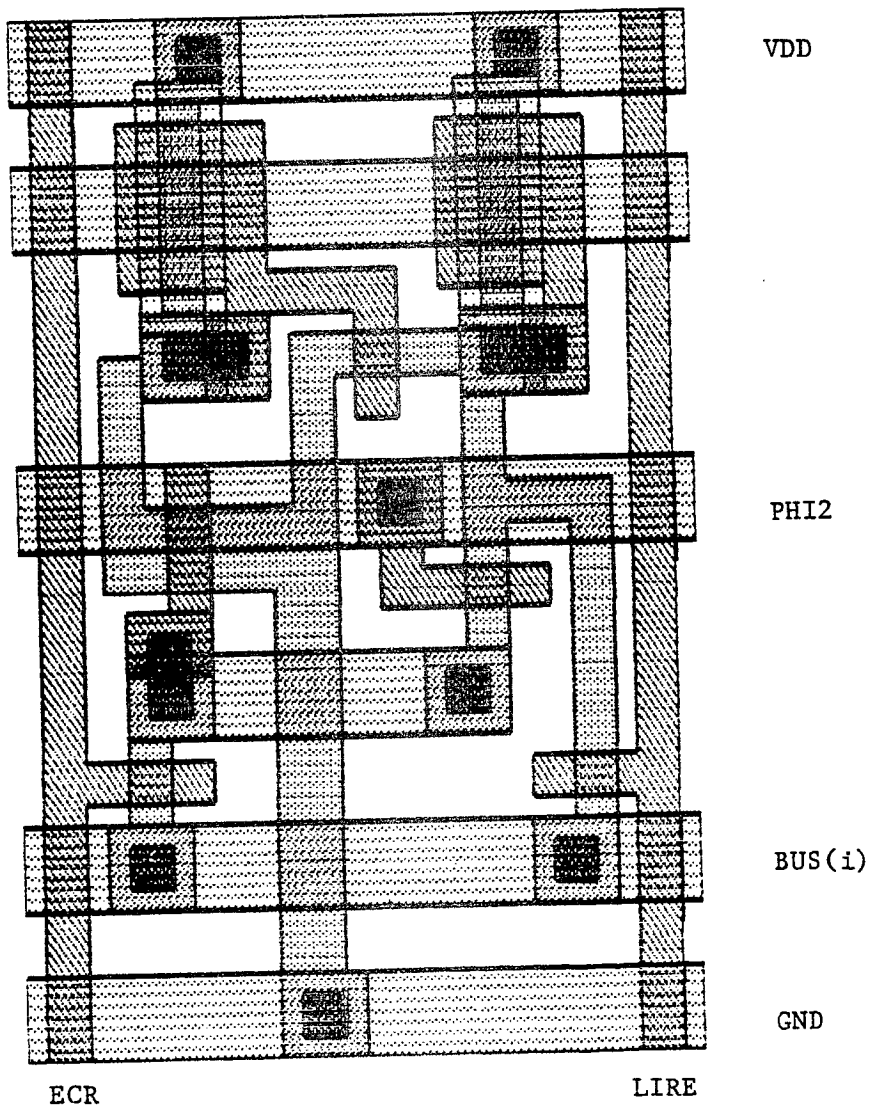


Figure 29: dessin d'une cellule registre general

Les commandes ont ete divisees en deux groupes suivant qu'elles sont actives pendant PHI1 ou PHI2. La taille d'une microinstruction est ainsi reduite a 1Ø bits, notes MØ a M9.

5.6.1.1-Microinstruction active pendant PHI1

Pendant PHI1, des transferts sur le bus general (BG) sont organises. La microinstruction executable pendant cette phase se compose de trois champs de 3, 3 et 4 bits respectivement comme l'indique la figure 3Ø.

ORIGINE			DESTINATION			REGISTRE			
M9	M8	M7	M6	M5	M4	M3	M2	M1	MØ

Figure 3Ø : Microinstruction active pendant PHI1

Le champ ORIGINE precise d'ou provient l'information a transferer sur BG. Le champ DESTINATION precise a quel endroit l'information doit parvenir. Dans le cas ou l'un des registres generaux intervient dans un transfert, l'adresse de ce registre est codee dans le champ REGISTRE. A noter que cette methode de codage interdit le transfert d'information entre registres generaux puisque l'adresse d'un seul registre peut etre codee. La figure 31 donne le codage des differentes commandes.

	codage
Champ ORIGINE	M9M8M7
- registre general	Ø Ø 1
- registre sortie UAL (Acc)	Ø 1 Ø
- sortie PLA	Ø 1 1
- registre memoire (RM)	1 Ø 1
- registre entree horizontale RH	1 1 Ø
- registre entree verticale RV	1 1 1
- rien	Ø Ø Ø
Champ DESTINATION	M6M5M4
- registre general	Ø Ø 1
- registre entree UAL (RA)	Ø 1 Ø
- registre entree PLA	Ø 1 1
- registre memoire (RM)	1 Ø 1
- registre sortie chip RS	1 1 Ø
- registre adresse memoire (RAM)	1 Ø Ø
- rien	Ø Ø Ø
Champ REGISTRE	
- nombre entre Ø et 15	

Figure 31: codage des champs de la microinstruction active pendant phil

5.6.1.2-Microinstruction active pendant PHI2

Les operations commandees par cette microinstruction sont de trois types: operations sur la memoire, operations sur l'UAL, operations d'entrees/sorties. La microinstruction est formee de trois champs de 2, 1, et 4 bits respectivement, comme l'indique la figure 32.

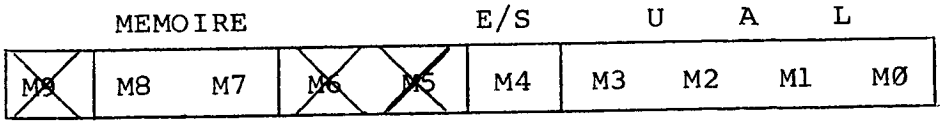


Figure 32 : microinstruction active pendant PHI2

On remarque que trois bits sont inutilises. La figure 33 donne le codage des differents champs de la microinstruction.

	codage M8	M7
Champ MEMOIRE		
- pas d'operation	Ø	Ø
- ecriture	Ø	1
- lecture	1	Ø
- lecture pour rafraichissement	1	1

	M4
Champ entree/sortie (E/S)	
- pas d'operation	Ø
- chargement registres RV et RH	1

Champ UAL	M3	M2	M1	MØ
- pas d'operation	Ø	-	-	-
- Ø --> Acc	1	Ø	Ø	Ø
- RA + Acc --> Acc	1	Ø	Ø	1
- RA - Acc --> Acc	1	Ø	1	Ø
- * Acc --> Acc	1	Ø	1	1
- Acc + 1 --> Acc	1	1	Ø	Ø
- RA * Acc --> Acc	1	1	Ø	1

Figure 33: codage de la microinstruction active pendant phi2

5.6.2-Decodage de la microinstruction

Rappelons qu'une microinstruction est composee de 1Ø bits, et que le chip recoit une nouvelle microinstruction a chaque phase de l'horloge. Suivant que la microinstruction s'execute pendant PHI1 ou PHI2, les champs de bits qu'elle comporte n'ont pas la meme signification. Le role du decodeur de microinstrustion est d'engendrer les commandes pour les differents modules en fonction d'une part de la phase courante de l'horloge et d'autre part, des valeurs des champs de la microinstruction.

Le decodage est effectue principalement dans 3 modules:
- Le decodeur de l'UAL
- Le decodeur d'adresse de registre
- Le decodeur des champs M4-M9

5.6.2.1-Decodage des operations sur l'UAL

Pour commander K, P, R et Cin en fonction de l'operation demandee, un PLA est utilise. De plus, afin de regrouper toutes les commandes concernant l'UAL dans un meme module, ce PLA va aussi engendrer deux autres commandes: la commande de chargement de l'accumulateur, active lors d'une operation sur l'UAL, et la commande de chargement du signe du resultat qui ne s'opere que sous certaines conditions. Le PLA

recoit 5 bits en entree: les bits MØ a M3 et la valeur du registre SIGNE. En combinant ces 5 informations, il genere les signaux K, P, R, Cin, CHbar/Sgn et CHbar/Acc. La figure 34 montre la table de verite du PLA.

M3	M2	M1	MØ	SGN	K3	K2	K1	KØ	P3	P2	P1	PØ	Cin	R3	R2	R1	RØ	CHbar /sgn	CHbar /acc
Ø	-	-	-	-	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	1
1	Ø	Ø	Ø	-	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
1	Ø	Ø	1	-	Ø	Ø	Ø	1	Ø	1	1	Ø	Ø	Ø	1	1	Ø	Ø	Ø
1	Ø	1	Ø	-	Ø	Ø	1	Ø	1	Ø	Ø	1	1	Ø	1	1	Ø	Ø	Ø
1	Ø	1	1	Ø	Ø	Ø	Ø	Ø	1	Ø	1	Ø	Ø	1	1	Ø	Ø	1	Ø
1	Ø	1	1	1	1	Ø	1	Ø	Ø	1	Ø	1	1	Ø	1	1	Ø	1	Ø
1	1	Ø	Ø	-	Ø	1	Ø	1	1	Ø	1	Ø	1	Ø	1	1	Ø	Ø	Ø
1	1	Ø	1	Ø	Ø	Ø	Ø	Ø	1	1	Ø	Ø	Ø	1	1	Ø	Ø	1	Ø
1	1	Ø	1	1	Ø	Ø	1	Ø	1	Ø	Ø	1	1	Ø	1	1	Ø	1	Ø

Figure 34: Table de verite du pla associe a l'UAL

Mise en oeuvre

Pour la realisation effective du PLA, quelques simplifications ont ete operees. On remarque en effet que RØ vaut toujours Ø et que K1 est egal a PØ. Ceci permet une legere diminution du nombre de sorties du PLA. Par ailleurs la condition d'entree 1ØØØ- ne donnant que des "Ø" en sortie, il est inutile de rentrer cette valeur effectivement dans le PLA, puisqu'un PLA fournit la valeur Ø en sortie, lorsqu'aucune condition d'entree n'est verifiee. Le PLA comporte finalement: 5 entrees, 13 sorties et 8 termes produits.

5.6.2.2-Decodage du champ registre

Le champ de microinstruction MØ a M3 permet le codage d'une valeur entre Ø et 15, et ainsi sert a selectionner l'un des 16 registres. Le decodeur d'operations sur un registre recoit les informations suivantes:
- 4 bits d'adresse permettant de selectionner l'un des 16 registres
- 1 bit de lecture
- 1 bit d'ecriture

La structure du decodeur est la meme que celle du decodeur memoire (cf figure 17).

5.6.2.3-Decodage des champs M4-M9

Le decodage du champ M4-M9 se fait dans un module appele DCDM49. Ce module est charge de generer, memoriser et envoyer des signaux de commandes aux differents modules. Le module DCDM49 est compose de deux parties comme le montre la figure 35. Une partie decodage (notee DCD49) qui reconnait certaines configurations binaires et une partie generation de signaux de commandes.

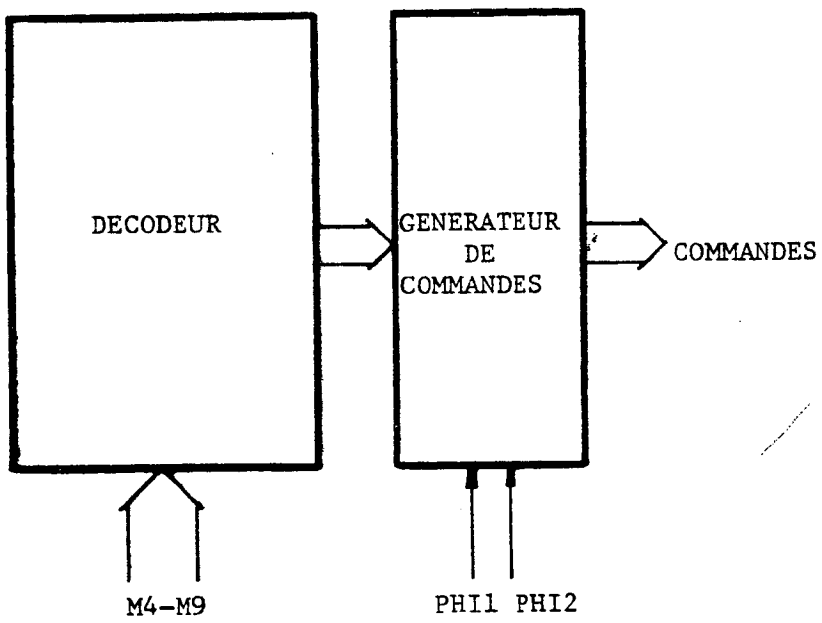


Figure 35: architecture du module DCDM49

Le decodeur est programme comme l'indique la figure 36.

M9	M8	M7	M6	M5	M4	SIGNAL ENGENDRE
Ø	Ø	1	-	-	-	LIRE REGISTRE
Ø	1	Ø	-	-	-	Acc/bus.PHI1
Ø	1	1	-	-	-	PLA/bus.PHI1
1	Ø	1	-	-	-	RM/bus.PHI1
1	1	Ø	-	-	-	RH/bus.PHI1
1	1	1	-	-	-	RV/bus.PHI1
-	-	-	Ø	Ø	1	ECR REGISTRE
-	-	-	Ø	1	Ø	CH/Acc.PHI1
-	-	-	Ø	1	1	CH/RPLA.PHI1
-	-	-	1	Ø	Ø	CH/RAM.PHI1
-	-	-	1	Ø	1	CH/RM.PHI1
-	-	-	1	1	Ø	CH/RS.PHI1
-	Ø	1	-	-	-	ECR MEMOIRE
-	Ø	1	-	-	-	ECR/MEM.PHI2
-	1	-	-	-	-	LIRE MEMOIRE
-	1	-	-	-	-	LIR/MEM.PHI2
-	1	1	-	-	-	REF/MEM.PHI2
-	-	-	-	-	1	CH/RH+RV.PHI2

Figure 36: Programmation du decodeur M49

Le generateur de signaux de commandes fournit des signaux de trois types:
- des signaux de commandes executables pendant PHI1. Ces signaux sont recus pendant la phase PHI2 et memorises pendant cette phase, puis envoyes pendant la phase PHI1 suivante (type 1).
- des signaux de commandes executables pendant PHI2. Ces signaux sont recus pendant la phase PHI1 et memorises pendant cette phase, puis envoyes pendant la phase PHI2 suivante (type 2).
- des signaux destines aux autres decodeurs. Ces signaux sont decodes et sont envoyes pendant le meme cycle (pas de memorisation) vers les decodeurs memoire et registre (type 3). Les signaux de ce troisieme type sont essentiellement les 4 signaux suivants:
  - lecture memoire
  - ecriture memoire
  - lecture registre
  - ecriture registre.

Mise en oeuvre

Les trois types de generateurs ont ete dessines de telle sorte qu'ils puissent s'empiler les uns sur les autres, partageant ainsi les signaux PHI1, PHI2, VDD, GND. La figure 37 represente les dessins des trois circuits et leur empilement.

5.7-Mise en place des modules sur la plaquette de silicium

L'une des phases les plus delicates dans la conception d'un microprocesseur consiste dans le placement, sur la plaquette de silicium, des modules qui le constituent. Cette phase est cruciale pour deux raisons principales. D'une part, elle determine la taille de la puce, facteur primordial actuellement pour la production de circuits

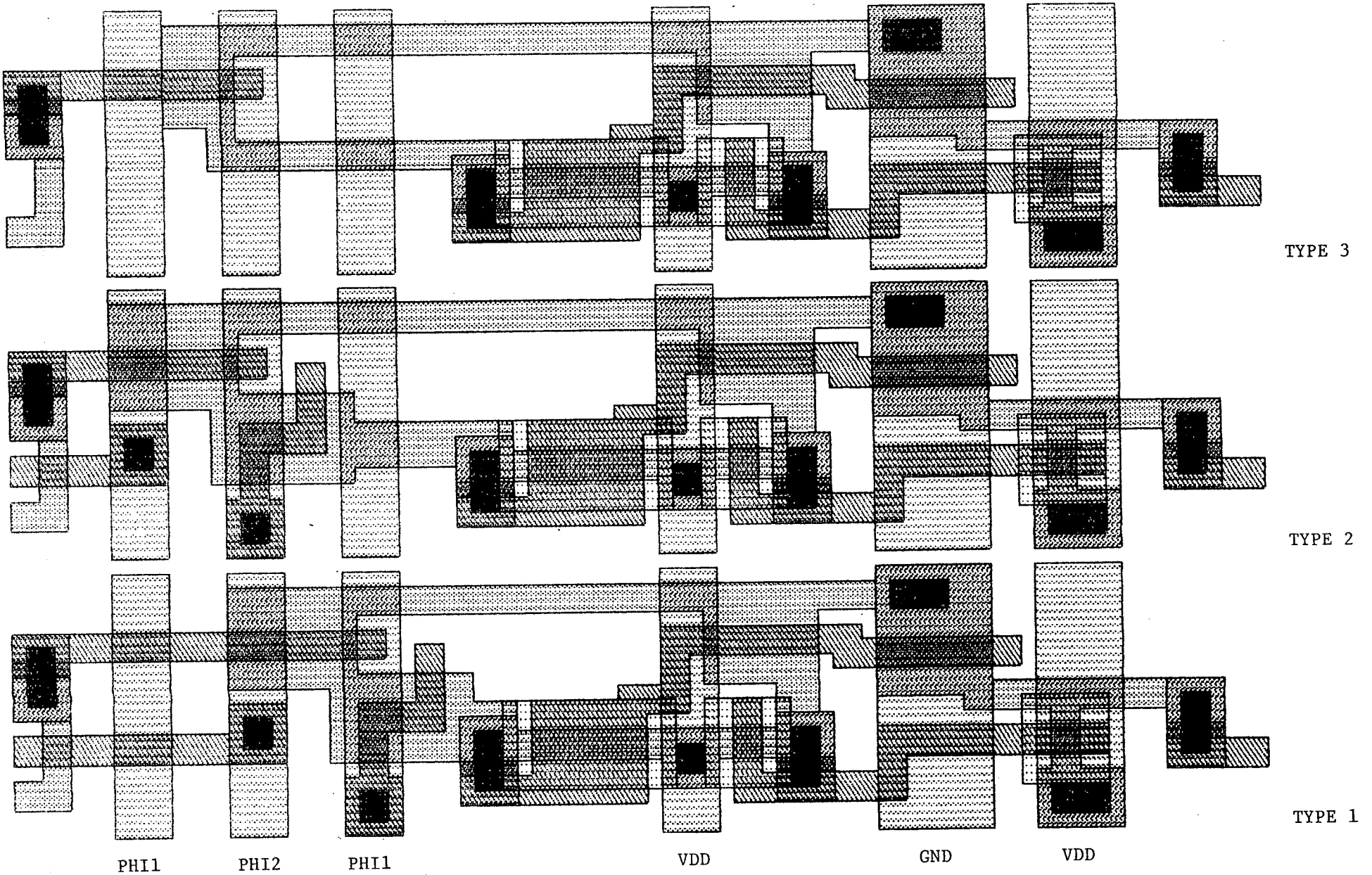


Figure 37: dessin des 3 types de generateurs de commandes

a grande echelle. En effet, plus le chip est grand, plus le rendement est faible et donc plus le circuit est cher. D'autre part, elle peut influencer les performances du processeur. En effet, une mauvaise repartition des modules entraine generalement l'utilisation de longs fils, d'un module a l'autre, d'ou une diminution des performances. Dans la technologie visee (NMOS 5 microns), la diminution de performances est peu sensible. Cependant, il faut conserver a l'esprit qu'avec les progres constants realises au niveau de l'integration, le temps de propagation des signaux sur des longs fils deviendra un facteur determinant dans les prochaines annees.

La figure 38 represente l'implantation des differents modules sur le silicium. Le chip comporte 62 plots de connexions, situes sur la peripherie de la puce, qui se repartissent de la maniere suivante:
- 3 series de 16 plots pour les entrees/sorties;
- 10 plots pour le controle des operations internes;
- 4 plots pour l'alimentation et les horloges.

Une version du chip avec seulement 18 plots de connexions etait envisageable en supposant que les entrees/sorties s'effectuent en serie. Par consequent, un plot par port d'entree/sortie etait necessaire. Plusieurs raisons ont motive notre choix pour la version parallele, notamment, la rapidite des tranferts, la facilite de programmation, de controle et de test du processeur, et egalement la simplicite de realisation. Il est clair, cependant, que la version serie doit etre envisagee dans le cas de l'integration de plusieurs processeurs sur la meme puce.

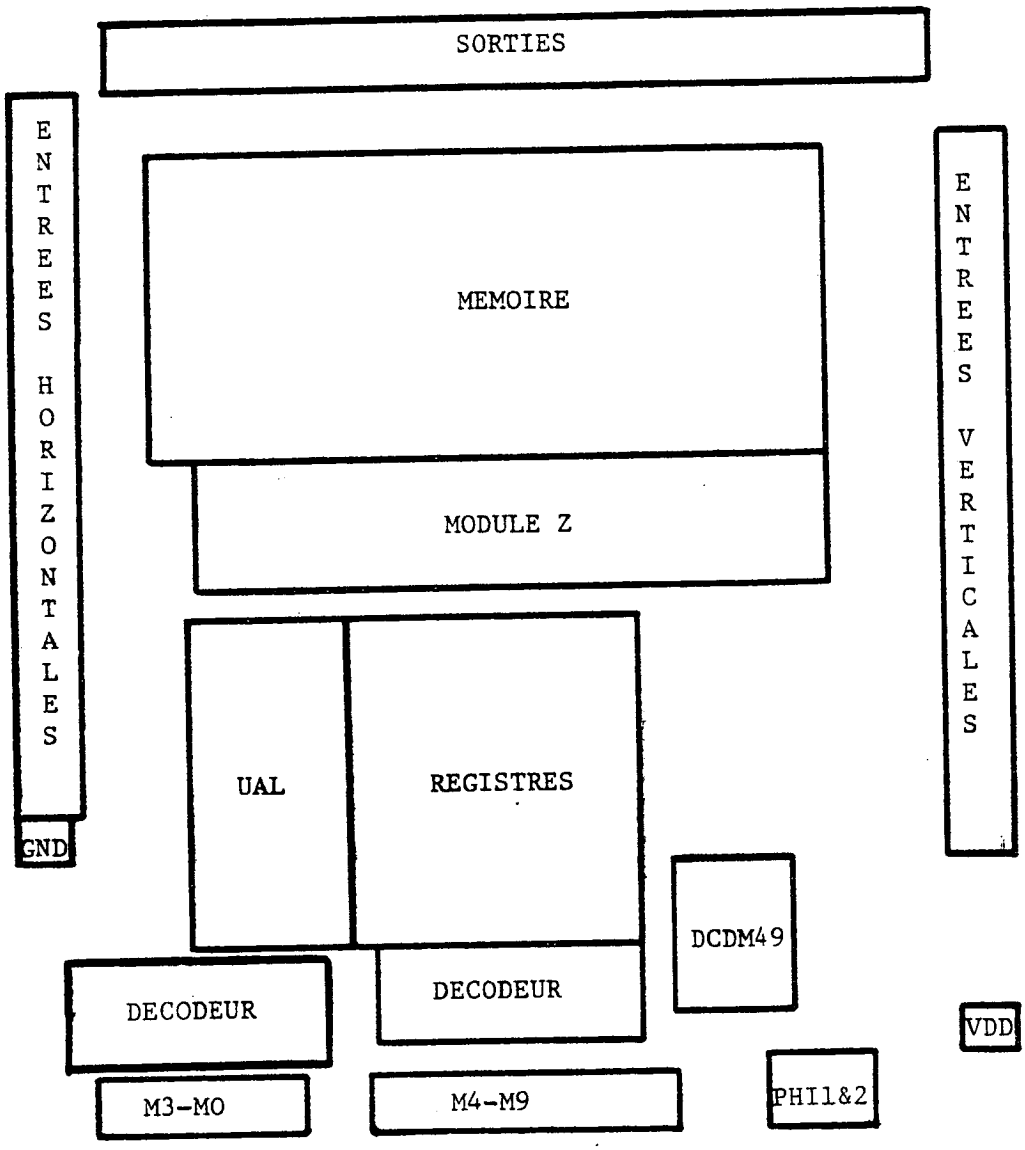
Les ports d'entrees horizontales et verticales du processeur se situent respectivement sur les cotes gauche et droit. Le port de sortie se trouve dans le haut. Le port de controle, recevant les microinstructions, ainsi que les plots des horloges PHI1 et PHI2, se situent dans le bas. La masse et l'alimentation occupent respectivement les plots inferieurs des cotes gauche et droit.

Le bus G est distribue sur 3 cotes du chip, le long des plots d'entrees/sorties du processeur. Ceci permet, d'une part, de connecter aisement les registres d'entrees/sorties (integres directement pres des plots correspondants) sur le bus et, d'autre part, de faciliter les connexions des differents modules sur le bus. Le bus G est constitue de 16 fils metalliques afin de minimiser les delais de transmissions d'informations.

Egalement dans le but de simplifier le probleme de cablage, les commandes destinees a la memoire, au module Z et aux registres d'entrees/sorties, sont regroupees sous forme d'un bus de 16 fils metalliques. Ce bus de controle s'etend sur le cote droit du chip, entre les bus G et les plots de connexions.

La memoire et le module Z occupent la partie superieure de la plaquette et sont connectes au bus G sur le cote droit. L'UAL et le tableau de registres sont places l'un a cote de l'autre, dans la partie inferieure du chip. Ils sont connectes au bus sur le cote gauche.

Les trois modules de decodage de microinstruction sont places dans le bas du circuit. Le decodeur d'operations arithmetiques est place


SORTIES
ENTREES HORIZONTALES
MEMOIRE
ENTREES VERTICALES
MODULE Z
UAL
REGISTRES
GND
DCDM49
DECODEUR
DECODEUR
VDD
M3-M0
M4-M9
PHI1&2

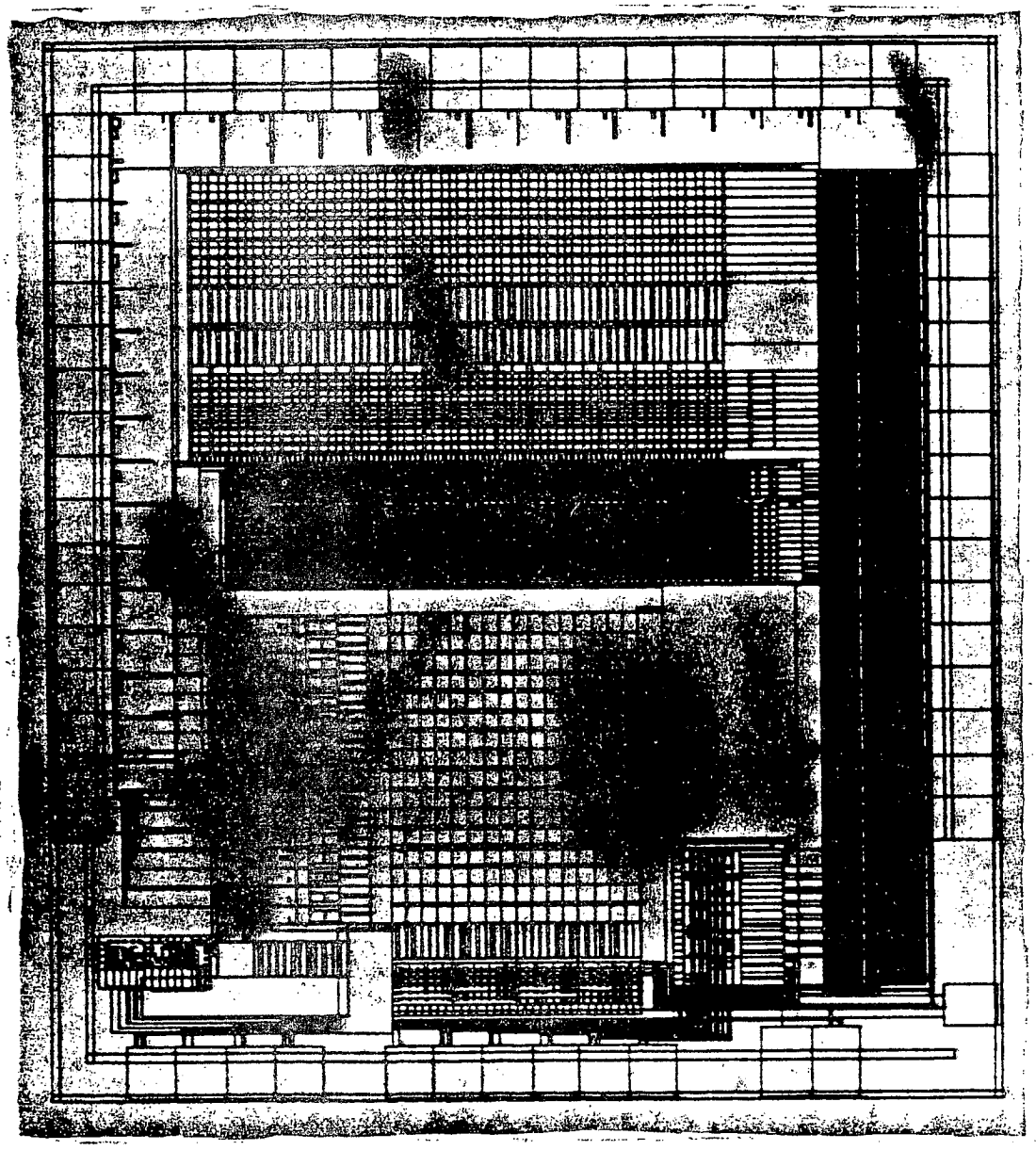


Figure 38: plan de la puce de silicium

sous l'UAL, le decodeur de registre, sous le tableau de registre. On note que les plots associes aux bits MØ-M3 de la microinstruction sont proches de ces deux decodeurs, reduisant ainsi la longueur de fils de connexions. Enfin, le decodeur du champ M4-M9 est place en bas, a droite, a cote du bus de commande.

6-Programmation du reseau de processeur

Dans cette partie, nous nous interessons a la programmation du tableau de processeurs ainsi qu'a ces performances. Avant le lancement du traitement en pipeline sur les mots, le multiprocesseur doit etre initialise. Cette initialisation comprend deux parties: d'une part le chargement des memoires des processeurs et d'autre part, le chargement de certains registres. Par ailleurs, la memoire etant dynamique, il est necessaire de prevoir un programme de rafraichissement qui s'executera periodiquement, toutes les 2 a 3 millisecondes, par exemple.

Dans un premier paragraphe, l'initialisation des memoires des processeurs est aborde. Puis le chargement des registres est etudie dans un second paragraphe. Dans le troisieme paragraphe, nous decrivons le programme de base d'un processeur. Enfin un programme de rafraichissement de la memoire des processeurs est donne dans le quatrieme paragraphe.

Dans la suite du texte, les programmes seront ecrits dans un pseudo langage de programmation proche de Pascal. En ce qui concerne les instructions d'entrees/sorties, nous utiliserons les 2 instructions suivantes: Afficher et Lire. L'instruction Afficher(X) signifie que la valeur X est envoyee sur le port de sortie. L'instruction Lire(E,X) est utilisee pour specifier que la valeur se trouvant sur le port d'entree E (Horizontal ou Vertical) doit etre rangee dans la variable X. Pour la plupart des programmes, les microprogrammes sont donnes en detail. Pour chaque phase d'horloge, nous donnons les commandes a activer.

6.1-Initialisation des memoires des processeurs

Nous avons vu que dans le cas du mode pipeline sur les mots, les memoires des processeurs doivent etre prealablement chargees avec les valeurs de probabilite de confusion qc(x/y), pour 3 valeurs de x et 20 valeurs de y (cf structure de la memoire dans le paragraphe 5.2.). Ces valeurs sont fournies sur le port d'entree verticale des processeurs de la premiere rangee. Les processeurs d'une meme colonne possedent les memes valeurs en memoire.

6.1.1-Chargement d'une colonne

Le principe du chargement des memoires des processeurs d'une meme colonne consiste a transferer les valeurs de processeurs en processeurs en mode pipeline. Pour ce faire, les valeurs a enregistrer sont precedees de leur adresse. Le fonctionnement est le suivant:

- a l'etape 0, seul le processeur <0,0> est actif. Il recoit successivement, sur son port d'entree verticale, les valeurs 0 et V(0) puis range V(0) a l'adresse 0.
- a l'etape 1, le processeur <0,0> recoit la valeur 1 et emet la valeur 0 vers le processeur <1,0>. Puis il recoit la valeur V(1) et transmet a <1,0> la valeur V(0). Le processeur <0,0> range la valeur V(1) a l'adresse 1 tandis que <1,0> range V(0) a l'adresse 0.
- plus generalement, a l'etape t, le processeur <j,0> recoit les valeurs t-j et V(t-j) et emet les valeurs t-j-1 et V(t-j-1) vers le processeur <j+1,0>.

6.1.2-Chargement de l'ensemble des processeurs

Les processeurs d'une meme diagonale etant commandes par un seul controleur, le chargement de la seconde colonne commence avec un cycle de retard par rapport a celui de la premiere colonne. Le chargement est represente sur la figure 39 et s'interprete ainsi:
- a l'etape Ø, seul le processeur <Ø,Ø> est actif. Le chargement des processeurs de la premiere colonne est lance.
- a l'etape 1, les processeurs <Ø,Ø>, <1,Ø> et <Ø,1> sont actifs. Le chargement des processeurs de la seconde colonne est lance.
- plus generalement, a l'etape t, les processeurs <i,j> tels que (i+j)<=t sont actifs. Le chargement des processeurs des colonnes 1 a t est en cours.

6.1.3-Programme de chargement

Le programme execute, a chaque etape, par un processeur actif est le suivant:

```
Debut
    Afficher (adresse);
    Lire (Vertical, adresse);
    Afficher (valeur);
    Lire (Vertical, valeur);
    memoire [adresse] := valeur
Fin
```

Programme 3: chargement des memoires

Ce programme s'explique ainsi. Afin que les processeurs se transmettent des valeurs, ils commencent par afficher la valeur a transmettre sur leur port de sortie, puis lisent la nouvelle valeur sur leur port d'entree verticale. Ce procede est repete pour la lecture de l'adresse et de la valeur. Puis le rangement memoire est effectue.

6.1.4-Temps de chargement des memoires

Le microprogramme correspondant au programme 3 precedent est le suivant:

```
1: REG(AD)/BUS ; CH/RS
2: CH/RV
1: RV/BUS ; CH/REG(AD)
2: -
1: REG(VAL)/BUS ; CH/RS
2: CH/RV
1: RV/BUS ; CH/REG(VAL)
2: -
1: REG(AD)/BUS ; CH/ADR
2: -
1: REG(VAL)/BUS ; CH/RM
2: ECR/MEM
```

microprogramme de chargement des memoires

Les registres notes AD et VAL sont 2 des 16 registres generaux. Le microprogramme est effectue en 6 cycles. Le temps de chargement de l'ensemble des memoires est egal au temps de chargement de la memoire

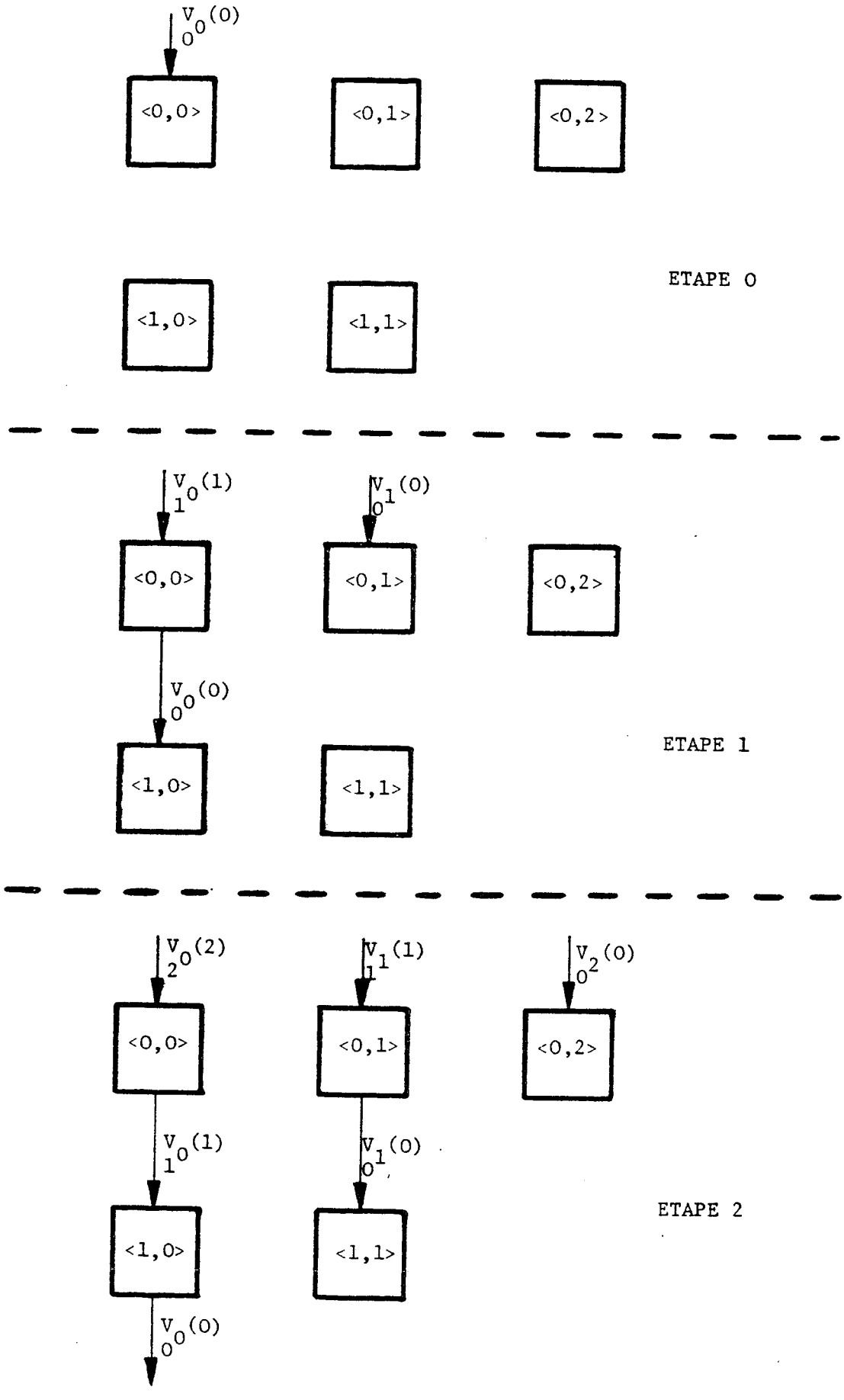


Figure 39: fonctionnement du reseau pour le chargement memoire

du processeur <Ø,Ø> auquel doit etre ajoute le temps de vidage du pipeline. Soient D le nombre de diagonales du systeme et E le temps d'execution du microprogramme (exprime en nombre de cycles). Le nombre de valeurs a charger par memoire etant de 6Ø, le nombre de cycles machines (note Cm) necessaire au chargement de toutes les memoires est donne par la formule:

$$Cm = 6Ø * E + (D-1) * E$$

ou encore

$$Cm = (D + 59) * E$$

A titre indicatif, avec E=6 et D=25, on obtient Cm=5Ø4 cycles.

6.2-Chargement des valeurs Xi et Qi(Xi)

Avant de lancer le traitement en pipeline sur les mots, les processeurs doivent enregistrer dans des registres des valeurs de deux types. D'une part, les valeurs de probabilites des 3 phonemes composant un segment Xi et d'autre part, les probabilites d'insertions associees. Comme dans le cas de la memoire, les processeurs d'une meme colonne possedent les memes informations. Cependant, le principe de chargement de ces valeurs se fait sur un mode different de celui du chargement des memoires vu dans le paragraphe precedent. Les valeurs etant a ranger dans des registres generaux, il n'est pas possible d'envoyer a un processeur une adresse, puis la valeur a ranger a cette adresse. En effet les registres ne sont adressables que par l'exterieur, notamment par les controleurs diagonaux (CD). Les CD fonctionnent de facon synchrone et engendrent les memes microinstructions ce qui signifie que les registres generaux doivent etre charges en parallele.

Le principe de fonctionnement adopte consiste, dans un premier temps, a diffuser la valeur a charger sur tous les ports d'entrees des processeurs d'une meme colonne en mode pipeline, puis dans un deuxieme temps, a diffuser a tous les processeurs une instruction de transfert de la valeur du port d'entree vers le registre adequat.

6.2.1-Programme de chargement des registres

Le programme de chargement est le suivant:

```
Debut
{ Diffusion d'une valeur sur une colonne de M processeurs }
  Faire M Fois
       afficher (valeur);
       lire (vertical, valeur)
  Fait;
{ Transfert de la valeur dans le registre R }
  R:=valeur
Fin
```

programme 4: Chargement des registres

Pendant l'execution de ce programme, tous les processeurs sont actifs. On suppose, de plus que la valeur a enregistrer est disponible pendant tout le temps de la boucle faire. De cette facon la valeur se propage de processeur en processeur. Lorsque la valeur a atteint le

dernier processeur d'une colonne, la commande de chargement dans un registre est envoyee.

6.2.2-Temps de chargement des registres

Le programme 4 precedent peut etre microprogramme de la facon suivante:

```
1: RV/BUS ; CH/RS      }   M
2: CH/RV               }  fois
1: RV/BUS ; CH/REG(R)
2: -
```

microprogramme de chargement des registres

Le nombre de valeurs a enregistrer est de 6. Le temps de chargement des registres, compte en nombre de cycles machines et note Cr, est donc donne par:

$$Cr = (M+1) * 6$$

Avec M=1Ø, Cc vaut 66 cycles.

6.3-Programmation d'un cycle processeur

Le programme execute par un processeur se decompose en 2 phases. D'une part, une phase de transmission d'information dans laquelle le processeur recoit des donnees de ces voisins et transmet des donnees a ces voisins. D'autre part, une phase de calculs dans laquelle le processeur determine les valeurs de Li, Lo et Lc.

6.3.1-Transmission d'information

Avant d'effectuer tout calcul, le processeur doit recevoir les informations suivantes:
- NY: nom du nouveau phoneme Y a traiter;
- YI,YC,YO: probabilites associees a Y;
- LI,LC,LO: resultats de processeurs voisins.

Par ailleurs, il doit envoyer les valeurs de meme type vers ces voisins. Le programme 5 est execute par tous les processeurs actifs pendant la phase de transmission.

```
Debut
   Afficher(LO); Lire(Vertical,LO);
   Afficher(NY); Lire(Horizontal,NY);
   Afficher(YI); Lire(Horizontal,YI);
   Afficher(YC); Lire(Horizontal,YC);
   Afficher(YO); Lire(Horizontal,YO);
   Afficher(LI); Lire(Horizontal,LI);
   Afficher(LC_EN_ATTENTE); Lire(Horizontal,NOUVEAU_LC);
   Afficher(LC); Lire(Vertical,LC_EN_ATTENTE); LC:=NOUVEAU_LC
Fin
```

Programme 5: transfert d'informations

Pour les valeurs LO, NY, YI, YC, YO et LI les transferts s'effectuent de facon simple: l'ancienne valeur est affichee sur le

port de sortie et la nouvelle valeur est lue sur l'un des ports d'entrees. Le cas de la valeur LC est plus complexe puisqu'un transfert diagonal doit etre emule. Il a ete vu que le transfert diagonal est remplace par 2 transferts successifs, notamment, un transfert vertical suivi d'un transfert horizontal (cf paragraphe 3.3.2). Un processeur contient donc a tout moment 2 valeurs de LC: d'une part, la valeur qu'il vient de calculer et d'autre part, une valeur LC (notee LC_EN_ATTENTE) en cours de tranfert diagonal, c'est a dire une valeur ayant effectue seulement un transfert horizontal. Pour obtenir la nouvelle valeur LC, le processeur recoit par transfert horizontal la valeur LC_EN_ATTENTE. L'ancienne valeur LC, quant a elle, est envoyee par transfert vertical dans la variable LC_EN_ATTENTE. Pour ce faire, la variable NOUVEAU_LC est utilisee temporairement.

6.3.2-Phase de calculs

Dans ce paragraphe, nous reprenons d'une facon plus precise le programme 2 decrit dans le paragraphe 4.1.2. En particulier, les instructions sont plus detaillees de facon a refleter la structure du processeur. La fonction SOMLOG, par exemple, qui calcule la valeur Log(a+b) en fonction de Log(a) et Log(b), est remplacee par une macro et est entierement programmee.

```
PROGRAMME CALCULS;
   MACRO SOMLOG(K1,K2);
{ cette macro range dans le registre K1 la valeur    }
{ suivante:      max(K1,K2) + Z(-abs(K1-K2))         }
     Debut
       K1:=K1-K2; SIGNE:=(K1>Ø);
       Si SIGNE
       Alors Debut
               K1:=-K1;
               K2:=K2 - K1;
             Fin;
       K1:=K2 + PLA(K1);
     Fin { Macro SOMLOG };

  Debut { Programme }
    { calcul de L }
    SOMLOG(Lo,Li);
    SOMLOG(Lo,Lc);
    L:= Lo;
    { calcul de Li }
    T1:=X1+I1;
    T2:=X2+I2;
    SOMLOG(T1,T2);
    T2:=X3+I3;
    SOMLOG(T1,T2);
    Li:=L+YI+T1;
    { calcul de Lc }
    Adr:=NY;
    T1:=X1+Qc[Adr];
    Adr:=Adr+1;
    T2:=X2+Qc[Adr];
    SOMLOG(T1,T2);
    Adr:=Adr+1;
    T2:=X3+Qc[Adr];
    SOMLOG(T1,T2);
    Lc:=L+YC+T1;
    { calcul de Lo }
    Lo:=L+Yo
  Fin { programme }
```

Programme 6: calculs

6.3.3-Temps d'execution

Les microprogrammes correspondants aux phases de transferts et de calculs sont donnes en annexe. On note que le nombre de cycles des phases de tranfert et de calcul, notes Ct et Cc respectivement, sont de 18 et de 87 cycles. Le temps total d'execution du programme de base, note Cp, est donc de 1Ø5 cycles.

6.4-Rafraichissement de la memoire

Periodiquement, la memoire des processeurs doit etre rafraichie. En effet, l'information est conservee sous forme de charge electrique sur la grille d'un transistor, et cette charge se dissipe apres quelques millisecondes.

6.4.1-Programme de rafraichissement

Le programme de rafraichissement consiste a lire chaque mot memoire puis a le reecrire. Nous avons vu dans la paragraphe que la memoire est structuree en 3 blocs de 20 elements. Une adresse memoire est formee par un doublet (A,B) ou A represente une adresse a l'interieur d'un bloc et B, le nuero du bloc. A appartient a l'intervalle [0,19] et est code sur 5 bits. B appartient a l'intervalle [0,2] et est code sur 2 bits. Le programme de rafraichissement consiste a balayer les adresses memoires entre 0 et 78 correspondants respectivement aux couples (A,B)=(0,0) et (A,B)=(19,2). Le programme 7 suivant procede au rafraichissement.

```
Debut
  Adr:=0;
  Faire 79 fois
    Debut
      Mem[Adr]:=Mem[Adr];
      Adr:=Adr+1
    Fin
Fin
```

Programme 7: rafraichissement des memoires

6.4.2-Temps d'execution

Le microprogramme associe au programme 7 est le suivant:

```
1: -
2: 0 --> ACC
1: ACC/BUS; CH/RAM            }  repeter
2: LIRE.MEM                   }  79
1: RM/BUS; CH/RM;             }  fois
2: ECR.MEM; ACC+1 --> ACC     }
```

microprogramme de rafraichissement

Ce microprogramme est execute simultanement sur tous les processeurs a la fin d'une phase de calculs par exemple. Le nombre de cycles necessaire au rafraichissement, note Cf, est de 159 cycles.

6.5-Estimation des performances du multiprocesseur

Lors du fonctionnement en mode pipeline sur les mots, le traitement a effectuer se decompose ainsi:
- chargement des memoires;
- chargement des registres;
- execution multiple du programme de base entrecoupee de rafraichissements memoire.

Ce traitement doit etre renouvele pour chaque suite de segments X(i),...,X(i+P). Si l'on desire detecter les mots d'un vocabulaire de d mots en temps reel, cela signifie que ce traitement devra s'effectuer pendant la duree de prononciation moyenne d'un phoneme. Nous allons estimer la taille de vocabulaire d que l'on peut esperer traiter. Trois hypotheses sont necessaires pour ce calcul:

- la duree moyenne d'un phoneme est estime a 1ØØ millisecondes;
- la duree d'un cycle d'horloge de la machine est estimee a environs 5ØØ nanosecondes (estimation grossiere et pessimiste);
- nous supposerons, enfin, que la memoire dynamique des processeurs doit etre rafraichie toutes les 2 millisecondes, soit 5Ø fois pendant le traitement d'un phoneme.

Nous noterons Tm, Tr, Tf, Tp, les temps d'executions associes aux valeurs Cm, Cr, Cf, Cp respectivement. Pendant la periode de 1ØØ millisecondes, le temps passe pour les initialisations et les rafraichissements (note Ti) est donne par la formule:

$$Ti = Tm + Tr + 5Ø \times Tf$$

soit Ti=4.3 milisecondes.

Soit Tu, le temps utilisable pour les calculs. Tu vaut 95.7 milisecondes. La taille du vocabulaire est donc donnee par:

$$d = Tu/Tp$$

soit environ 18ØØ mots. Cette valeur est tres nettement superieure a la valeur de 1ØØØ mots envisagee initialement.

Annexe du paragraphe 6

Dans cette annexe, nous decrivons les microprogrammes de transferts de donnees et de calculs executes pendant un cycle processeur.

Traduction du programme de transfert de donnees

```
1: REG[Lo]/Bus; CH/RS;
2: CH/RV;
1: RV/Bus; CH/REG[Lo];
2: -
1: REG[NY]/Bus; CH/RS;
2: CH/RH;
1: RH/Bus; CH/REG[NY];
2: -
1: REG[YI]/Bus; CH/RS;
2: CH/RH;
1: RH/Bus; CH/REG[YI];
2: -
1: REG[YC]/Bus; CH/RS;
2: CH/RH;
1: RH/Bus; CH/REG[YC];
2: -
1: REG[YO]/Bus; CH/RS;
2: CH/RH;
1: RH/Bus; CH/REG[YO];
2: -
1: REG[Li]/Bus; CH/RS;
2: CH/RH;
1: RH/Bus; CH/REG[Li];
2: -
1: REG[Lc_en_attente]/Bus; CH/RS;
2: CH/RH;
1: RH/Bus; CH/REG[nouveau_Lc];
2: -
1: REG[Lc]/Bus; CH/RS;
2: CH/RV;
1: RH/Bus; CH/REG[Lc_en_attente];
2: Ø --> Acc;
1: REG[nouveau_Lc]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[Lc];
2: -
```

nombre de cycles de la phase de TRANSFERT: 18

Traduction de la macro SOMLOG

```
2: Ø --> Acc;
1: REG[K2]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[K1]/Bus; CH/RA;
2: RA - Acc --> Acc;
1: Acc/Bus; CH/REG[K1];
2: * Acc --> Acc;
1: Acc/Bus; CH/REG[K1];
2: -
1: REG[K2]/Bus; CH/RA;
2: RA * Acc --> Acc;
1: REG[K1]/Bus; CH/RPLA;
2: -
1: PLA/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[K1];
```

nombre de cycles: 8

Dans les microprogrammes qui suivent, un registre note T2 est utilise. Le nombre de registres etant limite a 16, il s'agit en fait du meme registre que celui qui contient la valeur de Lo.

Traduction du calcul de L

```
1: -
SOMLOG(Lo,Li);
SOMLOG(Lo,Lc);
2: -
1: Acc/Bus; CH/REG[L];
```

nombre de cycles: 17

Traduction du calcul de Li

```
2: Ø --> Acc;
1: REG[X1]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[I1]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[T1];
2: Ø --> Acc;
1: REG[X2]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[I2]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[T2];
SOMLOG(T1,T2);
2: Ø --> Acc;
1: REG[X3]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[I3]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[T2];
SOMLOG(T1,T2);
2: -
1: REG[YI]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[L]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[Li];
```

nombre de cycles: 28

Traduction du calcul de Lc

Dans ce microprogramme, un registre note Adr est utilise. En fait, le nombre de registre etant limite a 16, ce registre est le meme que le registre Lc.

```
2: Ø --> Acc;
1: REG[NY]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[Adr];
2: -
1: REG[Adr]/Bus; CH/RAD;
2: LIRE/MEMOIRE; Ø --> Acc;
1: RM/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[X1]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[T1];
2: Ø --> Acc;
1: REG[Adr]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: -
2: Acc + 1 --> Acc;
1: Acc/Bus; CH/REG[Adr];
2: -
1: Acc/Bus; CH/RAD;
2: LIRE/MEMOIRE; Ø --> Acc;
1: RM/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[X2]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[T2];
SOMLOG(T1,T2);
2: Ø --> Acc;
1: REG[Adr]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: -
2: Acc + 1 --> Acc;
1: Acc/Bus; CH/RAD;
2: LIRE/MEMOIRE; Ø --> Acc;
1: RM/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[X3]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[T2];
SOMLOG(T1,T2);
2: -
1: REG[YC]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[L]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[Lc];
```

nombre de cycles: 38

Traduction du calcul de Lo

```
2: Ø --> Acc;
1: REG[YO]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[L]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[Lo];
2: -
```

nombre de cycles: 4

nombre de cycles de la phase CALCULS: 87

nombre total de cycles du programme de base d'un processeur: 1Ø5

Conclusion

Les travaux presentes dans ce rapport concernent la conception d'un circuit integre specialise pour une application de detection de mots dans la parole continue. Nous avons tout d'abord repris les travaux anterieurs et expose la methode de detection de mots puis l'algorithme parallele qui a ete imagine. Nous avons ensuite montre les differentes etapes qui ont abouti a la definition de la puce de silicium.

Dans une premiere etape, nous avons defini l'architecture du multiprocesseur capable de supporter l'algorithme parallele de detection. La machine est composee d'un tableau de processeurs interconnectes de facon reguliere et locale et fonctionnant de facon synchrone. Chaque processeur possede deux ports d'entrees et un port de sortie. Le controle de l'ensemble est realise grace a un controleur general qui diffuse un flot d'instruction vers les processeurs. Ces instructions transitent par des controleurs diagonaux qui gerent une diagonale de processeurs.

Dans une seconde etape, nous avons defini les fonctions devant etre realisees par chaque processeur. L'algorithme que chaque processeur doit derouler nous a permis d'inventorier les fonctions qu'il doit realiser. Ces fonctions sont de deux ordres: d'une part, des fonctions de calculs et d'autre part, des fonctions de memorisation. Les calculs a effectuer sont des calculs de probabilite faisant intervenir des sommes de produits. Le passage aux logarithmes a permis de simplifier les operations a effectuer qui se reduisent a des additions, soustractions et consultation de table. En ce qui concerne les fonctions de memorisation du processeur, elles ont ete reduites au minimum puisque chaque processeur contient un memoire de soixante mots et un ensemble de 16 registres.

Dans une troisieme etape, l'architecture d'un processeur a ete definie. Le processeur est constitue principalement de 4 modules: un module memoire ou sont stockees des valeurs de probabilites de confusions entre phonemes; une unite arithmetique et logique capable d'effectuer des operations elementaires telles que addition, soustraction, incrementation ; un tableau de registres servant de memoire de travail; un module note Z, utilise pour tabuler une fonction.

Dans une quatrieme etape, les differents modules constituants le processeur ont ete abordes et les circuits associes concus. Les masques des circuits ont ete dessines ensuite, sachant que les circuits seraient realises dans une technologie NMOS 5 microns. Le circuit definitif s'integre sur une puce de silicium de 5mm sur 6mm et comporte environ 12000 transistors. Il comporte 62 plots d'entrees/sorties qui se decomposent en 3 groupes de 16 plots pour les ports d'entrees/sorties proprement dits, 10 plots recevant les microinstructions et 4 plots pour l'alimentation, la masse et les signaux d'horloge.

Dans une derniere partie, nous nous sommes interesses aux performances du processeur. Nous avons montre comment initialiser le multiprocesseur puis comment le programmer. Les differents microprogrammes ont ete donnes ce qui nous a permis d'evaluer les performances de la machine. Nous avons montre en particulier, que la machine etait capable de traiter le probleme de la detection de mots en temps reel sur un vocabulaire d'environ 1800 mots alors que l'objectif initial etait de traiter environ 1000 mots.

Perspectives

La methodologie de conception de circuits integres preconisee par Mead et Conway, semble prometteuse. Si l'on s'en tient a notre propre experience, 6 semaines de cours suivis de 8 mois de travail ont permis le dessin d'un circuit integre d'environ 12000 transistors. Les performances estimees permettent d'envisager un fonctionnement des circuits sur un vocabulaire de pres de 2 milles mots, alors que nous avions table sur un vocabulaire d'un millier de mots.

Le circuit a ete dessine en visant une technologie NMOS a 5 microns. Cette technologie est loin d'etre la plus sophistiquee puisqu'au niveau industriel une technologie de moins de 3 microns est couramment utilisee. Par ailleurs, au niveau de la recherche, on est descendu deja au dessous du micron. Cela signifie qu'il est desormais possible d'integrer sur une meme puce de silicium plusieurs processeurs comme celui decrit dans ce rapport. A moyen terme, il est clair que le multiprocesseur complet pourra etre integre. Cependant, 2 problemes importants concernant une telle integration meritent d'etre soulignes. D'une part, la consommation du circuit risque d'etre enorme et une technologie de type CMOS, beaucoup moins consommatrice de courant, doit etre consideree. D'autre part, l'integration du multiprocesseur complet pose le probleme du nombre de plots d'entrees/sorties. Une facon de limiter ce nombre consiste a effectuer des entrees/sorties en mode serie.

A plus court terme, notre etude va se poursuivre dans 2 domaines principaux. D'une part, avant de fabriquer le circuit integre, nous nous proposons d'effectuer des simulations du circuit a deux niveaux: niveau logique et niveau electrique, afin de verifier le fonctionnement logique du circuit et d'estimer de facon assez precise, ses performances. D'autre part, nous envisageons le fabrication du circuit et son test.

Bibliographie

1. Kung H.T., Why Systolic Architectures?, Computer vol 15, no 1, Janvier 1982, pp 37-46.

2. Mercier G., Nouhen A., Quinton P., Siroux J., The Keal Speech Understanding System, in Spoken Language Generation and Understanding, J.C. Simon Editor, Proceedings of the NATO ASI, Bonas, 1980.

3. Banatre J.P., Frison P., Quinton P., A Network for the Detection of Words in Continuous Speech, VLSI Int Conf, J.P. Gray Editor, Academic Press, 1981.

4. Bahl L.R., Jelinek F., Decoding for Channels with Insertions, Deletions, and Substitutions with Application to Speech Recognition, IEEE Trans. on Information Theory, 21, no 4, Juillet 1976, pp 404-411.

5. Landman H.A., MAKE PLA Users' Guide, ERL, University of California, Berkeley, 1981.

6. Mead C., Conway L., Introduction to VLSI Systems, Addison-Wesley, 1980.

7. Hon R.W., Sequin C.H., A Guide to LSI Implementation, Xerox PARC Technical Report SSL-79-7, Palo Alto, California, 1980.

Liste des Publications Internes IRISA

PI 150 **Construction automatique et évaluation d'un graphe d'«implication» issu de données binaires, dans le cadre de la didactique des mathématiques**
H. Rostam , 112 pages ; Juin 1981

PI 151 **Réalisation d'un outil d'évaluation de mécanismes de détection de pannes]-]Projet Pilote SURF**
B. Decouty, G. Michel, C. Wagner, Y. Crouzet , 59 pages ; Juillet 1981

PI 152 **Règle maximale**
J. Pellaumail , 18 pages ; Septembre 1981

PI 153 **Corrélation partielle dans le cas « qualitatif »**
I.C. Lerman , 125 pages ; Octobre 1981

PI 154 **Stability analysis of adt ptively controlled not-necessarily minimum phase systems with disturbances**
Cl. Samson , 40 pages ; Octobre 1981

PI 155 **Analyses d'opinions d'instituteurs à l'égard de l'appropriation des nombres naturels par les élèves de cycle préparatoire**
R. Gras , 37 pages ; Octobre 1981

PI 156 **Récursion induction principle revisited**
G. Boudol, L. Kott , 49 pages ; Décembre 1981

PI 157 **Loi d'une variable aléatoire à valeur R^+ réalisant le minimum des moments d'ordre supérieur à deux lorsque les deux premiers sont fixés**
M.Kowalowka, R. Marie , 8 pages ; Décembre 1981

PI 158 **Réalisations stochastiques de signaux non stationnaires, et identification sur un seul échantillon**
A. Benveniste J.J. Fuchs , 33 pages ; Mars 1982

PI 159 **Méthode d'interprétation d'une classification hiérarchique d'attributs-modalites pour l'«explication» d'une variable ; application à la recherche de seuil critique de la tension artérielle systolique et des indicateurs de risque cardiovasculaire**
B. Tallur , 34 pages ; Janvier 1982

PI 160 **Probabilite stationnaire d'un réseau de files d'attente multiclasse à serveur central et à routages dépendant de l'état**
L.M. Le Ny , 18 pages ; Janvier 1982

PI 161 **Détection séqentielle de changements brusques des caractéristiques spectrales d'un signal numérique**
M. Basseville, A. Benveniste , pages ; Mars 1982

PI 162 **Actes regroupés des journées de Classification de Toulouse (Mai 1980), et de Nancy (Juin 1981)**
I.C. Lerman , 304 pages ;

PI 163 **Modélisation et Identification des caractéristiques d'une structure vibratoire : un problème de réalisation stochastique d'un grand système non stationnaire**
M. Prévosto, A. Benveniste, B. Barnouin , 46 pages ; Mars 182

PI 164 **An enlarged definition and complete axiomatization of observational congruence of finite processes**
Ph. Darondeau , 45 pages ; Avril 1982

PI 165 **Accès vidéotex à une banque de données médicales**
A. Chauffaut, M. Dragone, R. Rivoire, J.M. Roger , 25 pages ; Mai 1982

PI 166 **Comparaison de groupes de variables définies sur le même ensemble d'individus**
B. Escofier, J. Pages , 115 pages ; Mai 1982

PI 167 **Transport en circuits virtuels internes sur réseau local et connexion Transpac**
M. Tournois, R. Trépos , 90 pages ; Mai 1982

PI 168 **Impact de l'intégration sur le traitement automatique de la parole**
P. Quinton , 14 pages ; Mai 1982

PI 169 **A systolic algorithm for connected word recognition**
J.P. Banâtre, P. Frison, P. Quinton , 13 pages ; Mai 1982

PI 170 **A network for the detection of words in continuous speech**
J.P. Banâtre, P. Frison, P. Quinton , 24 pages ; Mai 1982

PI 171 **Le langage ADA : Etude bibliographique**
J. André, Y. Jégou, M. Raynal , 12 pages ; Juin 1982

PI 172 **Comparaison de groupes de variables : 2ème partie : un exemple d'application**
B. Escofier, J. Pajès , 37 pages ; Juillet 1982

PI 173 **Unfold-fold program transformations**
L. Kott , 29pages ; Juillet 1982

PI 174 **Remarques sur les langages de parenthèses**
J.M. Autebert, J. Beauquier, L. Boasson, G. Senizergues , 20 pages ; Juillet 1982

PI 175 **Langages de parenthèses, langages N.T.S. et homomorphismes inverses**
J.M. Autebert, L. Boasson, G. Senizergues , 26 pages ; Juillet 1982

PI 176 **Tris pour machines synchrones ou Baudet Stevenson revisited**
R. Rannou , 26 pages ; Juillet 1982

PI 177 **Un nouvel algorithme de classification hiérarchique des éléments constitutifs de tableau de contingence basé sur la corrélation**
B. Tallur , Juillet 1982 ;

PI 178 **Programmes d'analyse des re'sultats d'une classification automatique**
I.C. Lerman et collaborateurs , 79 pages ; Septembre 1982

PI 179 **Attitude à l'égard des mathématiques des élèves de sixième**
J.Degouys, R. Gras, M. Postic , 29 pages ; Septembre 1982

PI 180 **Traitements de textes et manipulations de documents : bibliographie analytique**
J. André , 20 pages ; Septembre 1982

PI 181 **Algorithme assurant l'insertion dynamique d'un processeur autour d'un réseau à diffusion et garantissant la cohérence d'un système de numérotation des paquets global et réparti**
Annick Le Coz, Hervé Le Goff, Michel Ollivier , 31 pages ; Octobre 1982

PI 182 **Interprétation non linéaire d'un coefficient d'association entre modalités d'une juxtaposition de tables de contingence**
Israël César Lerman , 34 pages ; Novembre 1982

PI 183 **L'IRISA vu à travers les stages effectués par ses étudiants de DEA (1è année de thèse)**
Daniel Herman , 41 pages ; Novembre 1982

PI 184 **Commande non linéaire robuste des robots manipulateurs**
Claude Samson , 52 pages ; Janvier 1983

PI 185 **Dialogue et représentation des informations dans un système de messagerie intelligent**
Philippe Besnard, René Quiniou, Patrice Quinton, Patrick Saint-Dizier, Jacques Siroux, Laurent Trilling , 45 pages ; Janvier 1983

PI 186 **Analyse classificatoire d'une correspondance multiple ; typologie et régression**
I.C. Lerman , 54 pages ; Janvier 1983

PI 187 **Estimation de mouvement dans une sequence d'images de télévision en vue d'un codage avec compensation de mouvement**
Claude Labit , 132 pages ; Janvier 1983

PI 188 **Conception et réalisation d'un logiciel de saisie et restitution de cartes élémentaires**
Eric Sécher , 45 pages ; Janvier 1983

PI 189 **Etude comparative d'algorithmes pour l'amélioration de dessins au trait sur surfaces point par point**
M.A. ROY , 96 pages ; Janvier 1983

PI 190 **Généralisation de l'analyse des correspondances à la comparaison de tableaux de fréquence**
Brigitte Escofier , 35 pages ; Mars 1983

PI 191 **Association entre variables qualitatives ordinales «nettes» ou «floues»**
Israël-César Lerman , 42 pages ; Mars 1983

PI 192 **Un processeur intégré pour la reconnaissance de la parole**
Patrice Frison , 80 pages ; Mars 1983

PI 193

PI 194 **Régime stationnaire pour une file M/H/1 avec impatience**
Raymond Marie et Jean Pellaumail , 8 pages ; Mars 1983

PI 195 **SIGNAL : un langage pour le traitement du signal**
Paul Le Guernic, Albert Benveniste, Thierry Gautier , 49 pages ; Mars 1983

PI 196 **Algorithmes systoliques : de la théorie à la pratique**
Françoise André, Patrice Frison, Patrice Quinton , 19 pages ; Mars 1983

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique